TRABAJO DE FINAL DE CARRERA

TÍTULO DEL TFC: Aplicaciones didácticas de PLD/FPGA para las asignaturas de sistemas digitales

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Sistemas de Telecomunicaciones / Ingeniería Técnica de Aeronáutica, especialidad Aeronavegación

AUTOR: Jonatan González Rodríguez

DIRECTOR: Francesc Josep Sánchez i Robert

Fecha: 24 de marzo de 2012

Título: Aplicaciones didácticas de PLD/FPGA para las asignaturas de sistemas digitales

Autor: Jonatan González Rodríguez

Director: Francesc Josep Sánchez i Robert

Fecha: 24 de marzo de 2012

Resumen

Este proyecto es una guía destinada a los futuros alumnos de la asignatura CSD que se imparte en el grado en Ingeniería de Sistemas de Telecomunicaciones en la EETAC y para personas que se inicien en el lenguaje de programación VHDL.

En el presente trabajo de final de carrera se parte desde cero en este tipo de lenguaje para facilitar su comprensión a nuevos usuarios del mismo.

En el capítulo 2 se explica cómo crear un proyecto desde el principio, compilarlo, simularlo y verlo de manera empírica sobre una placa.

En el capítulo 3 se hace un ejemplo de un circuito combinacional de dos maneras y en el capitulo 4 uno secuencial.

El capitulo 5 se centra en el diseño de un UART sencillo para transmitir y recibir datos simultáneamente con un PC a través del puerto serie.

En el capitulo 6 se resume un proyecto de antiguos alumnos que aprovechamos para implementarlo junto con el UART en el capitulo 7.

Y finalmente, en el capitulo 8 se hace una interfaz gráfica con Labview para ver la comunicación entre la placa y el PC.

Todos los documentos usados, los códigos, las simulaciones y los ejercicios están en la página web de la asignatura.

Title: Educational applications of PLD / FPGA for digital systems courses

Author: Jonatan González Rodríguez

Director: Francesc Josep Sánchez i Robert

Date: March, 24th 2012

Overview

This bachelor thesis is a guide and tutorial for future students of the subject Digital Circuits and Systems (CSD) in the Telecommunications Engineering bachelor degree at the EETAC. It also can be of reference for any professional searching for materials to learn VHDL and programmable logic devices. In this final project we start from scratch in VHDL in order to facilitate its comprehension to new users.

In chapter 2, we explain how to create a new project from the beginning using Xilinx ISE tools, compile it, simulate it using the integrated ISim simulator, and how to synthesise and implement it in a training and demonstration board from Xilinx. RTL view and technology views of circuit will also be presented.

In chapter 3, we perform an example of a combinational circuit in design styles: behavioural and using hierarchical components. In chapter 4, we perform again the design procedure using the same strategies and tools, but focusing on sequential systems.

In chapter 5, we focus on the design of a dedicated processor, a simple UART intended to send and receive serially a byte of data at a given transmission baud-rate and protocol. The subsystem can be used to connect the training board to a computer COM port.

In chapter 6, the previous UART project is enhanced and instantiated together with another tutorial project, a traffic light controller. The aim is to use a Labview based graphic computer interface to monitor and control the activity in the street intersection.

All the learning materials such as documents, VHDL code, presentation slides and final ISE projects, will be available on-line presented in a Google Sites web, which will be freely accessible through a link in the CSD web.

A Francesc, tutor de este proyecto, por guiarme durante el mismo y darme la oportunidad de aprender.

A mis padres, Elisa y Jesús, por su apoyo y ánimos para tener los estudios que yo quería.

A mi hermano, Joaquín, por su apoyo durante toda la carrera y en especial durante la realización de este proyecto.

A Marta, por sus ánimos en todo, especialmente estos dos últimos años.

Lo que sabemos es una gota de agua,
lo que ignoramos es el océano.
(Isaac Newton)

INTRODUCCIÓN

Este proyecto sirve como guía a los futuros alumnos de la asignatura *Digital Circuits and Systems (CSD).* Con este proyecto se pretende que los alumnos tengan una referencia para empezar a programar en VHDL y, para ello, se ha creado una página web donde se han subido algunos ejercicios resueltos a modo de ejemplo como introducción a la programación en este tipo de lenguaje.

Mi punto de partida en este proyecto es la asignatura de *Electrónica Digital (ED)* donde no se empleó VHDL, pero si s*e realizaron circuitos digitales en programas tales como Proteus.*

Primero veremos un ejemplo de cómo funciona el programa *ISEProject Navigation,* que es el que se utiliza para programar la placa *Spartan 3E-500* de *Xilinx,* haciendo un circuito sencillo como es encender un LED a través de un interruptor, y seguidamente veremos un ejemplo de un sistema combinacional.

Después haremos un ejemplo de un sistema secuencial y cómo unir varios circuitos en un mismo proyecto.

Y finalmente diseñaremos un UART y estableceremos comunicación entre la placa Spartan 3E-500 y el PC a través del puerto serie. Para hacer esta comunicación más visual y dinámica realizaremos otro proyecto en el cual intervendrá *Labview,* una herramienta gráfica que se puede comunicar a través del puerto serie.

Todos estos ejemplos tienen unos pasos muy importantes a seguir siempre: especificaciones, diagrama de estados y/o de tiempos, diseño del circuito, simulación y pruebas en la placa.

La simulación la haremos mediante el *ModelSim* que lleva integrado el propio *ISEProject Navigation* de *Xilinx.*

1 INTRODUCCIÓN A VHDL

VHDL es la combinación de VHSIC (*Very High Speed Integrated Circuit*), y HDL, (*Hardware Description Language)*.

VHDL sirve para describir circuitos digitales complejos. Esto supone una ventaja ya que por ejemplo, mediante otros métodos como diagramas de bloques, ya no resulta práctico de realizar.

Otros lenguajes que hacen lo mismo son *Verilog* y *ABEL.*

Este lenguaje se usa principalmente para programar PLD (*Programable Logic Device*) y FPGA (*Field Programable Gate Array*), como es nuestro caso.

En VHDL hay diferentes maneras de describir un circuito, habiendo que decidir qué manera es la más adecuada para cada uno:

- Funcional (behavioural): Se describe la forma en la que se comporta el circuito. La descripción es secuencial.
- Estructural: Se describe el circuito de forma jerárquica, sabiendo en todo momento como quedará el circuito físico.
- Mixta: una mezcla de las dos anteriores.

También existen otras maneras de diseñar los circuitos como son los filtros digitales, bancos de pruebas o maquinas de estados, como veremos en alguno de los ejercicios más adelante.

2 PRIMER EJERCICIO EN VHDL

En este capitulo aprenderemos a usar el ISE Project Navigator. Lo primero que haremos será crear un proyecto nuevo, después aprenderemos a insertar un código en VHDL, a continuación veremos como hacer una simulación para comprobar que el circuito diseñado cumple con las especificaciones iniciales y finalmente aprenderemos a asignar las entradas/salidas de nuestro circuito a los pines de la placa e insertar el código en ella.

Para crear un proyecto tenemos que ir a *File-> New Project.*

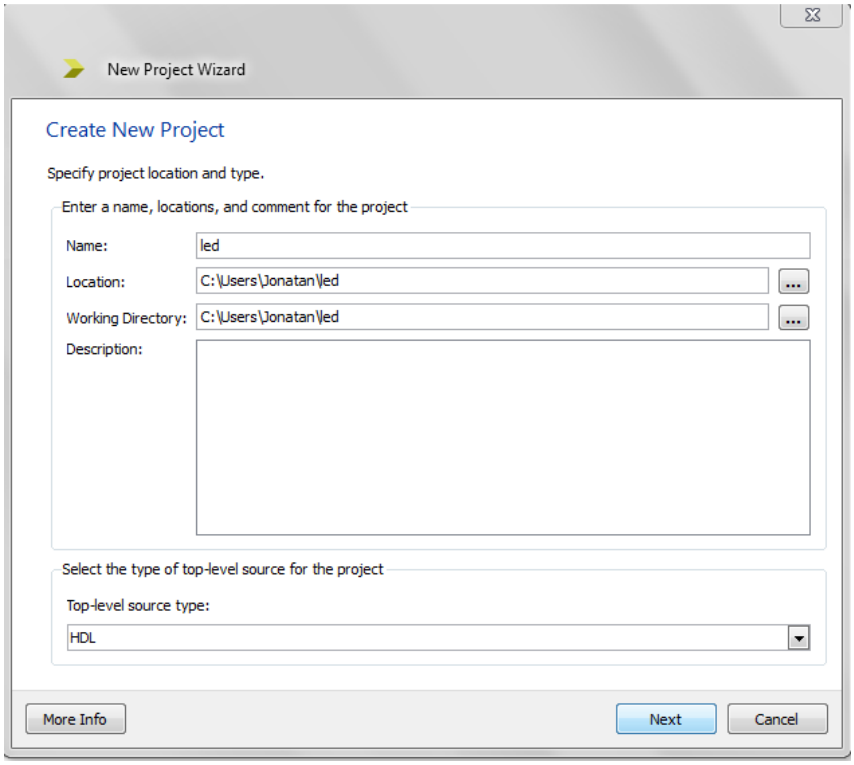


Figura 1 Creación de un proyecto

En la ventana que se abre hay que introducir el nombre del proyecto, el directorio donde queremos que se guarde y el tipo de fuente, en nuestro caso HDL y al clickar en *next* saldrá otra ventana.

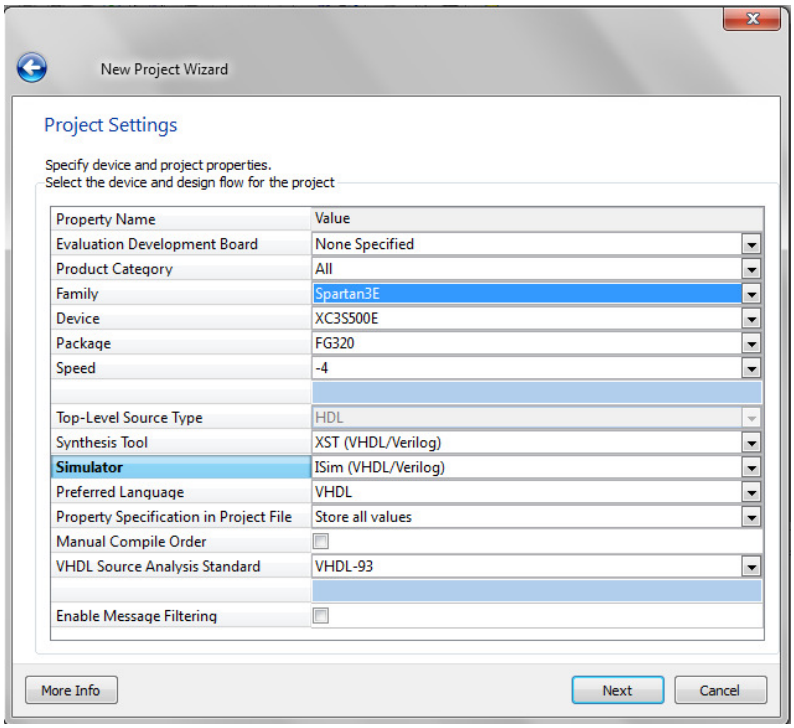


Figura 2 Creación de un proyecto

En esta ventana tenemos que escoger la FPGA que vayamos a usar, en nuestro caso *Spartan3E XC3S500E FG320*. También es importante escoger el simulador, nosotros trabajaremos con el simulador que lleva integrado Xilinx, el *ISim(VHDL/Verilog).* La siguiente ventana es un resumen de los parámetros que hemos elegido.

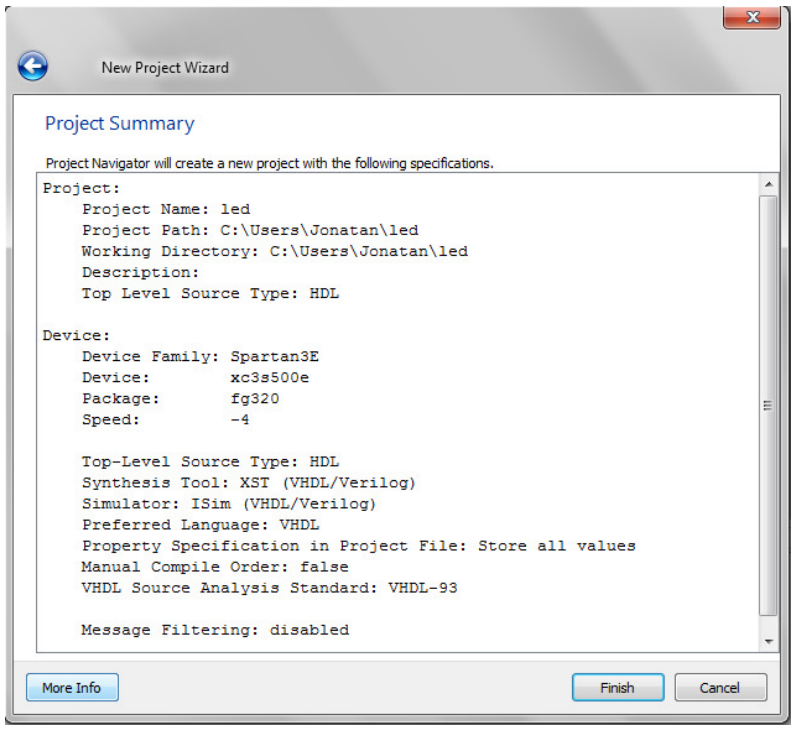


Figura 3 Creación de un proyecto

2.1. Encender un LED

En este primer ejercicio haremos algo sencillo como es encender un LED para ver como funciona el programa y el lenguaje que tratamos de aprender.

Para ello seguiremos paso a paso todo lo que hay que hacer. Lo primero que tenemos que hacer es crear un proyecto como hemos explicado en el capítulo anterior.

A continuación tenemos que definir unas especificaciones que el circuito deberá cumplir, una vez hecho esto ya podemos hacer el circuito describiéndolo en VHDL dentro del proyecto. Cuando esté compilado ya podremos simularlo con el simulador que lleva integrado *Xilinx*, *ISim.* Al comprobar que cumple con las especificaciones ya podemos insertarlo en la FPGA.

2.1.1. Especificaciones

Este ejemplo constará de una entrada y una salida. La entrada será un interruptor y la salida será un LED.

Las especificaciones en este caso son sencillas: cuando el interruptor esté activado el LED se encenderá.

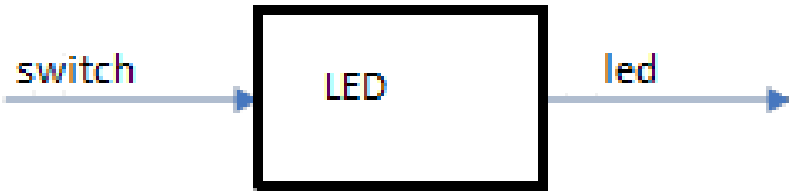


Figura 4 Especificaciones

2.1.2. Código

Para introducir un código tenemos que ir a *New Source*, que está marcado en rojo en la siguiente figura.

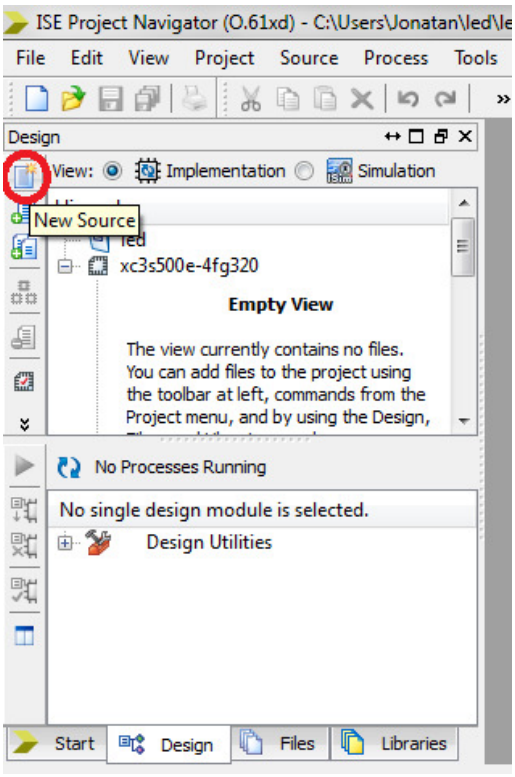


Figura 5 Instrucciones para insertar un código nuevo

Al hacer click sobre New Source nos aparece la siguiente ventana.

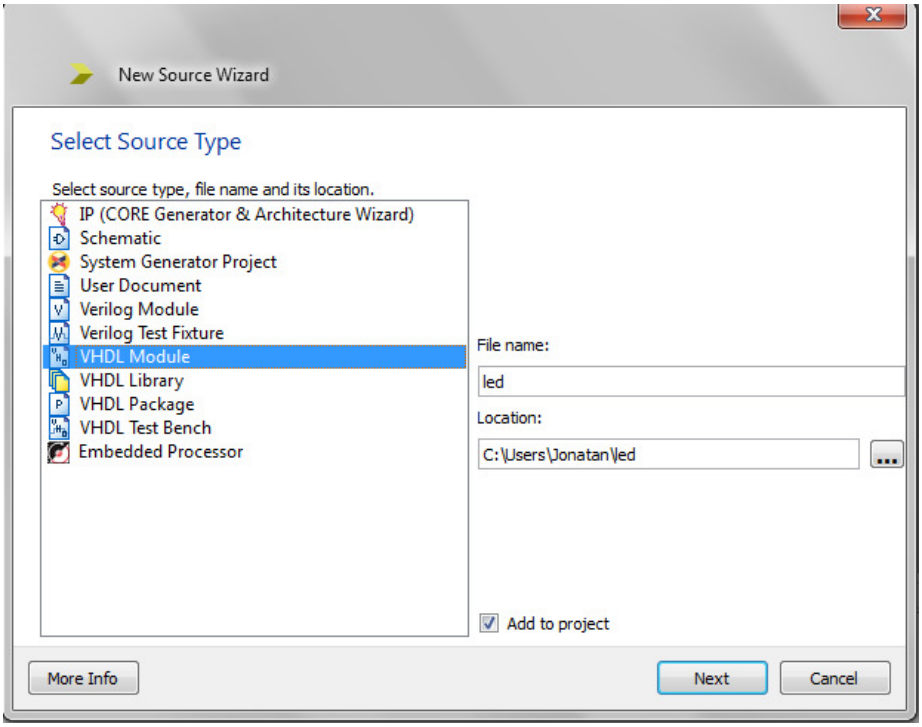


Figura 6 Instrucciones para insertar un código nuevo

En esta ventana tenemos que poner el nombre del código que vayamos a hacer e indicar que es del tipo *VHDL Module.* Es importante que la casilla *Add to Project* esté activada para que se añada a nuestro proyecto.

A continuación nos aparece una ventana donde están resumidos todos los parámetros escogidos y tenemos que indicar *Finish.*

Una vez hecho esto ya podemos escribir el código.

El código a usar para encender un LED con un interruptor es el siguiente.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Led is
port(

        switch:         in std_logic;
        led:            out std_logic

        );

end Led;

architecture Behavioral of Led is

begin

        led <= switch;

end Behavioral;
```

2.1.3. Código RTL

Una vez tenemos el circuito descrito el VHDL, Xilinx nos ofrece ver el circuito que hemos descrito de forma gráfica. Para hacerlo tenemos que seguir los siguientes pasos.

Lo primero de todo es guardar el circuito descrito en VHDL, a continuación tenemos que hacer doble click en *Synthesize* y esperar a que en la ventana inferior salga el mensaje *“Process "Synthesize - XST" completed successfully”.* Es en está ventana donde se informa de los posibles errores de código que podamos tener. Una vez hemos hecho esto tenemos que hacer doble click en *View RLT Schematic.*

En la siguiente figura podemos ver marcados los pasos que tenemos que seguir.

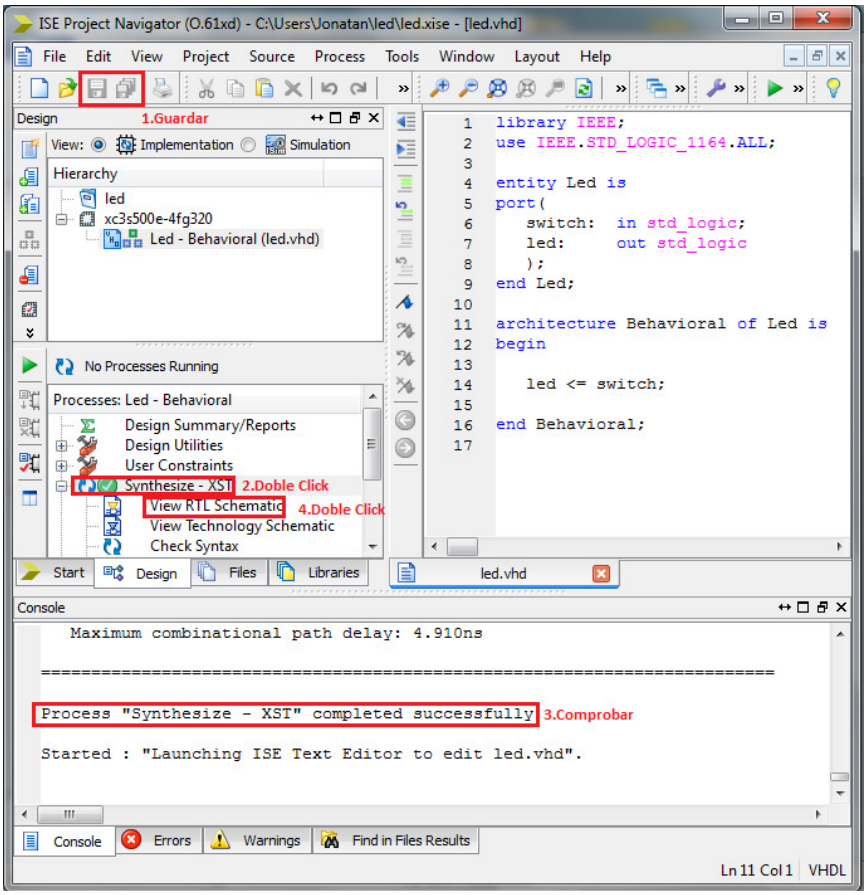


Figura 7 Crear Circuito RTL

A continuación sale la ventana siguiente.

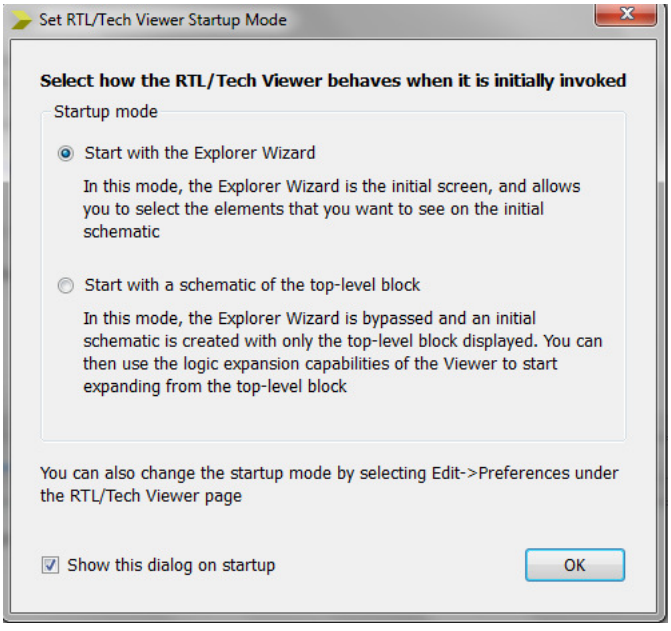


Figura 8 Crear circuito RTL

En la primera opción nos aparece un menú donde podemos elegir que bloque queremos ver en el circuito y en la segunda opción nos aparece directamente

el bloque principal. En este caso es indiferente ya que es un ejercicio con un único bloque sencillo.

El circuito es el siguiente.

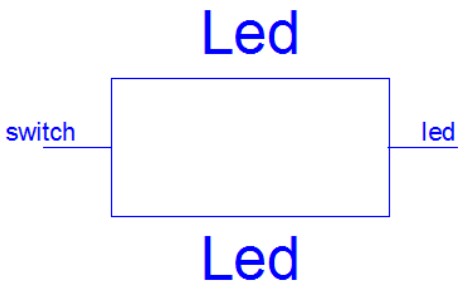


Figura 9 Circuito RTL

Se puede entrar dentro del circuito haciendo doble click en cualquier bloque, en este ejercicio solo hay un cable, esta opción será de gran utilidad en ejercicios más complejos.

2.1.4. Simulación

La simulación la haremos con el *ModelSim* que lleva integrado *Xilinx* en la versión 13.2. Para ello hay que crear un *testbench*, el cual nos va a permitir ver si realmente el circuito descrito en VHDL funciona como nosotros lo hemos diseñado.

Para introducir el código de simulación tenemos que ir a *Simulation*. Una vez allí tenemos que ir a *New Source.*

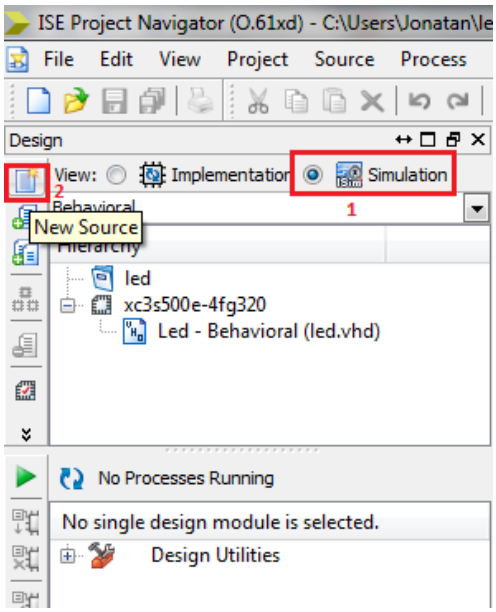


Figura 10 Simulación

Una vez hecho esto nos saldrá la siguiente ventana.

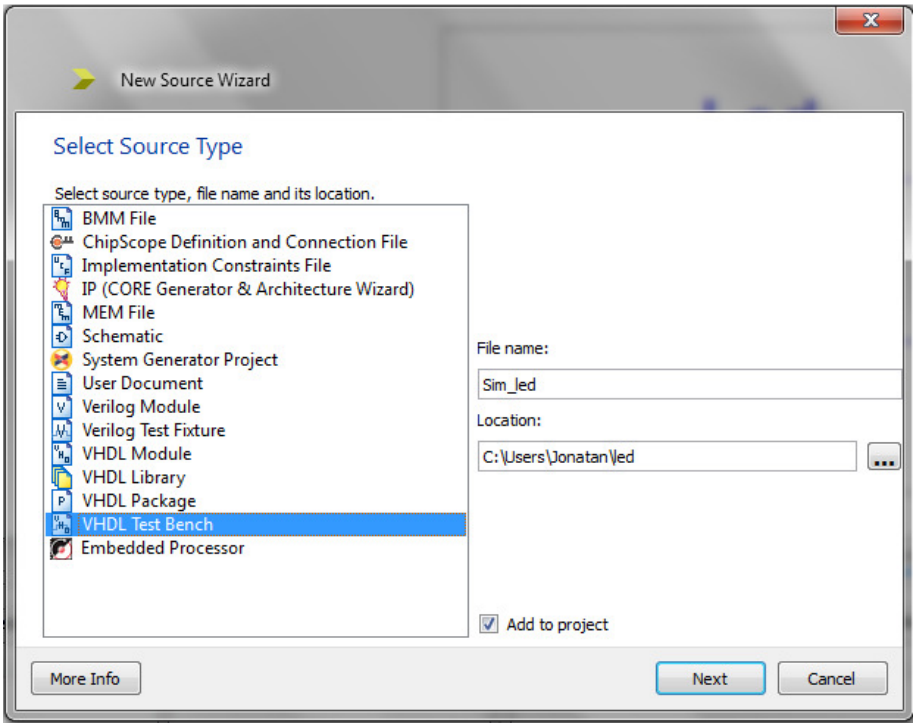


Figura 11 Simulación

Esta vez tenemos que escoger *VHDL Test Bench* e introducir un nombre al archivo de simulación. En la siguiente ventana tenemos que escoger sobre que bloque queremos hacer la simulación, en nuestro caso solo tenemos uno.

A continuación ya podemos escribir el código, el cuerpo del cual ya está prediseñado, únicamente tenemos que introducir código donde indica *Insert stimulus here.* El código que se introduce es el cambio que queremos hacer en las entradas del circuito.

El código entero de la simulación es el siguiente:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Sim_led IS
END Sim_led;

ARCHITECTURE behavior OF Sim_led IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Led
    PORT(
         switch : IN  std_logic;
         led : OUT  std_logic
        );
    END COMPONENT;

   --Inputs
   signal switch_signal : std_logic := '0';

       --Outputs
```

```
   signal led_signal : std_logic;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: Led PORT MAP (
          switch => switch_signal,
          led => led_signal
        );

   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
      -- insert stimulus here
               switch_signal <= '1';
               wait for 10 us;
               switch_signal <= '0';
               wait for 20 us;
               switch_signal <= '1';
               wait for 50 us;
               switch_signal <= '0';
               wait for 25 us;
               switch_signal <= '1';
               wait for 30 us;
               switch_signal <= '0';
               wait for 30 us;
      wait;
   end process;

END;
```

Para iniciar la simulación tenemos que hacer doble click en *Behavioral Check Syntax,* esperar a que en la ventana inferior aparezca "*Process "Behavioral Check Syntax" completed successfully"* y a continuación hacer doble click en *Simulate Behavioural Model.*

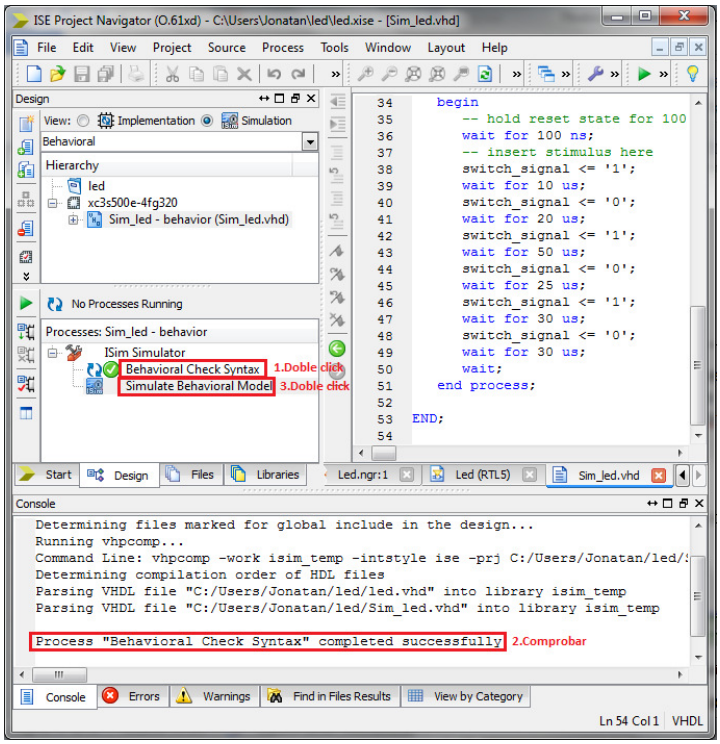


Figura 12 Simulación

A continuación aparece una ventana donde están las señales de entrada y salida de nuestro circuito. Para iniciar la simulación tenemos que clicar únicamente en *play* y ajustar los ejes de la simulación para comprobar lo que queremos.

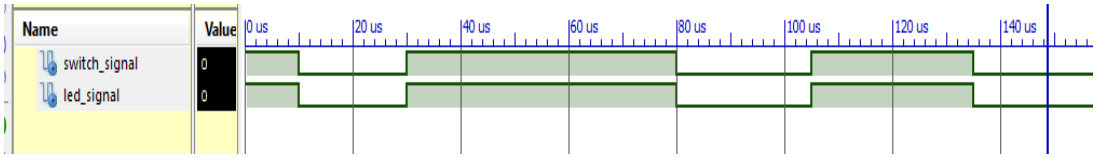


Figura 13 Simulación en el *ModelSim*

Con esta simulación podemos ver que cuando se activa el interruptor el LED se enciende.

2.1.5. Implementación en la placa

Para insertar el circuito diseñado en la placa *Spartan 3E-500* hace falta un código *.ucf* donde se asignan los puertos de entrada y de salida del circuito a los de la placa física. En este caso es el siguiente.

```
NET "switch" LOC = G18;
NET "led" LOC = J14;
```

El código se genera automáticamente al asignar las entradas/salidas del circuito a los pines de la placa.

Para hacer estas asignaciones tenemos que hacer doble click en *I/O Pin Planning (PlanAhead).*

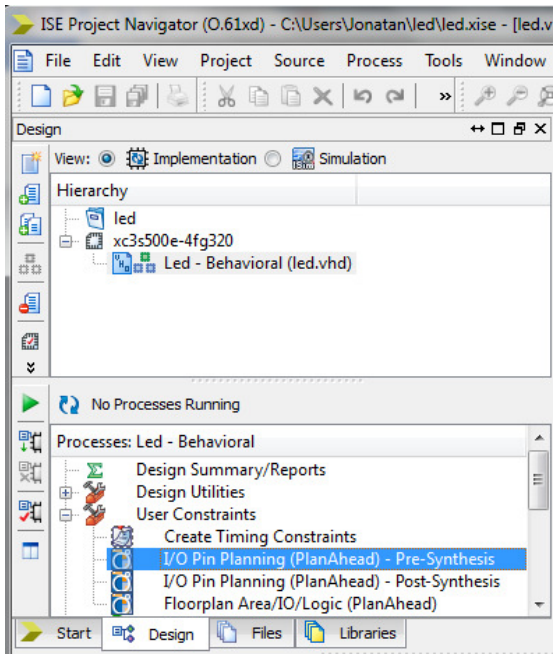


Figura 14 Asignación de pines

A continuación nos aparece una ventana donde podemos asignar las entradas/salidas de nuestro circuito con los pines de la placa.

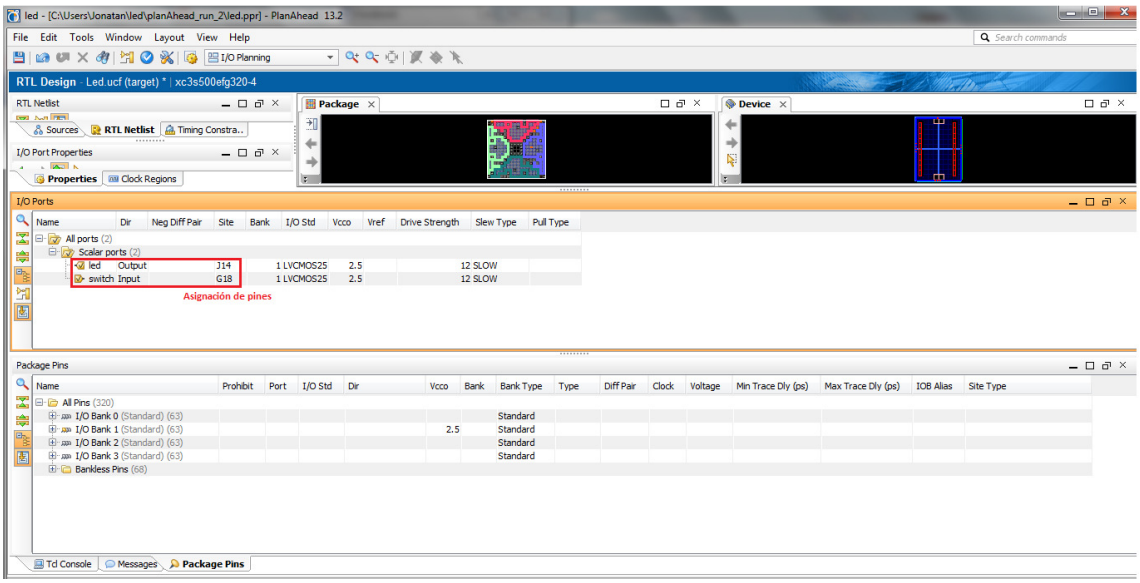
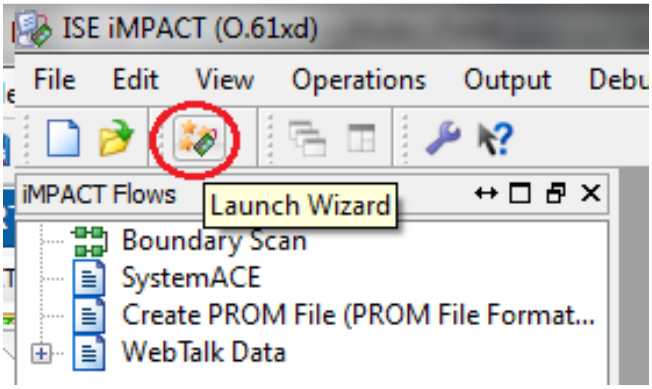


Figura 15 Asignación de pines

Una vez tenemos asignados los pines solo nos queda insertar en la FPGA el circuito descrito.

Para hacer esto tenemos que hacer doble click en *Generate Programming File,* y si no hay errores ir a *Configure Target Device -> Manage Configuration Project (iMPACT).* Se abrirá una ventana donde tenemos que ir a *Launch Wizard.*



Seremos direccionados a otra ventana donde tenemos que indicar que queremos que la placa se detecte automáticamente.

A continuación nos preguntan si queremos asignar alguna archivo e indicamos que si y asignamos el archivo que se ha generado automáticamente con extensión *.bit.*

Para acabar de programar la placa tenemos que ir con el botón derecho encima de la imagen que dice *Xilinx* e indicar *Program.*

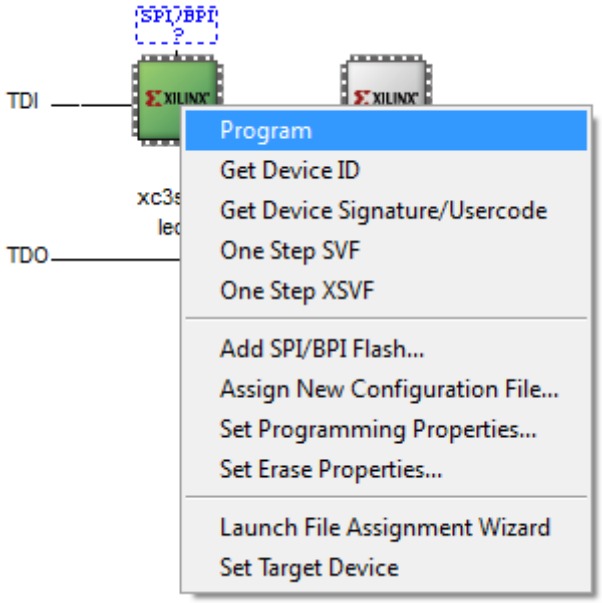


Figura 16 Programar la FPGA

Una vez programada la FPGA ya podemos comprobar en la placa que el circuito cumple con las especificaciones iniciales.

3 CIRCUITOS COMBINACIONALES

Los circuitos combinacionales son aquellos cuyas salidas sólo dependen de las entradas. Para comprender un ejemplo de estos circuitos haremos un codificador de 8 a 3.

3.1. Especificaciones

Nuestro ejercicio consta de nueve entradas, de las cuales 8 son pulsadores (I0..I7) y una es un pulsador. Lo hacemos así porque la placa no tiene más pulsadores. Y de 5 salidas que son LEDS. A continuación se muestra el bloque con sus entradas y salidas.

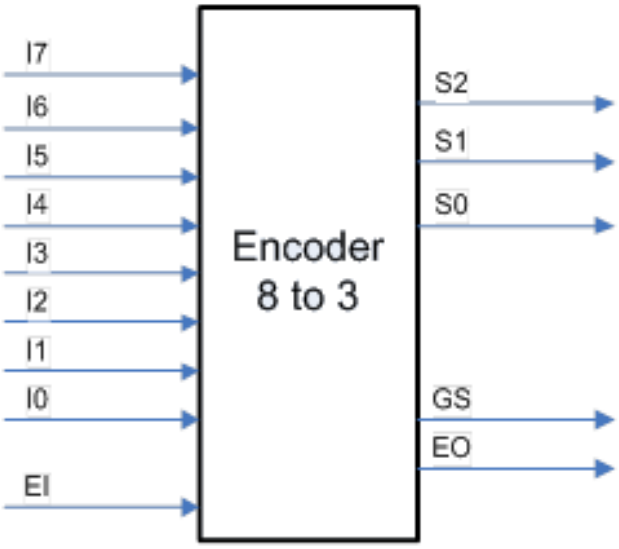


Figura 17 Especificaciones del codificador 8 a 3

Las especificaciones que tiene el circuito se resumen en la tabla de la verdad, que se muestra a continuación.

Entradas									Salidas				
EI	I7	I6	I5	I4	I3	I2	I1	I0	S2	S1	S0	EO	GS
0	X	X	X	X	X	X	X	X	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	X	0	0	1	0	1
1	0	0	0	0	0	1	X	X	0	1	0	0	1
1	0	0	0	0	1	X	X	X	0	1	1	0	1
1	0	0	0	1	X	X	X	X	1	0	0	0	1
1	0	0	1	X	X	X	X	X	1	0	1	0	1
1	0	1	X	X	X	X	X	X	1	1	0	0	1
1	1	X	X	X	X	X	X	X	1	1	1	0	1

Tabla 1 Tabla de la verdad

3.2. Descripción del circuito en VHDL

A partir de aquí ya podemos empezar a programar el circuito en VHDL. Lo haremos de dos maneras diferentes, funcional y estructural, y veremos que las dos cumplen con las especificaciones iniciales.

3.2.1. Descripción funcional

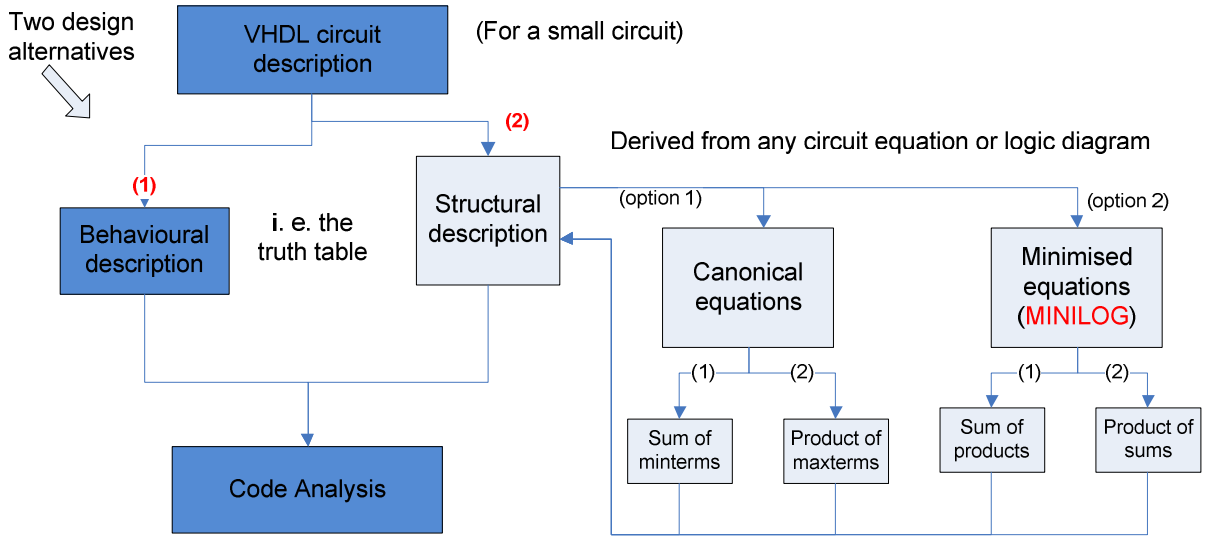


Figura 18 Descripción funcional

En esta opción simplemente hay que introducir en código VHDL la tabla de la verdad de nuestro codificador, la cual se encuentra en las especificaciones. El programa hace el circuito de tal manera que obtiene el resultado final sin que podamos controlar nada del circuito interno.

Un ejemplo de insertar esta tabla de la verdad en VHDL es el siguiente:

```
library IEEE;
use     IEEE.STD_LOGIC_1164.all;

entity encoder_8_to_3 is

                port (
                        I7,I6,I5,I4,I3,I2,I1,I0,EI: in STD_LOGIC;

                        EO,GS,S0,S1,S2: out STD_LOGIC
                          );

end encoder_8_to_3;

architecture Tabla_Verdad of encoder_8_to_3 is
                        signal V_ENT: STD_LOGIC_VECTOR(8 downto 0);
                        signal V_SAL: STD_LOGIC_VECTOR(4 downto 0);
begin
                        V_ENT <= EI & I7 & I6 & I5 & I4 & I3 & I2 & I1 & I0;

                        S2 <= V_SAL(4);
                        S1 <= V_SAL(3);
                        S0 <= V_SAL(2);
```

```
                    GS <= V_SAL(1);
                    EO <= V_SAL(0);

                    with V_ENT select

                                V_SAL <=        "00000" when "0--------",
                                                "00010" when "100000000",
                                                "00001" when "100000001",
                                                "00101" when "10000001-",

                                                "01001" when "1000001--",
                                                "01101" when "100001---",
                                                "10001" when "10001----",
                                                "10101" when "1001-----",
                                                "11001" when "101------",
                                                "11101" when "11-------",
                                                "11111" when others;

end Tabla_Verdad;
```

3.2.2. Descripción estructural

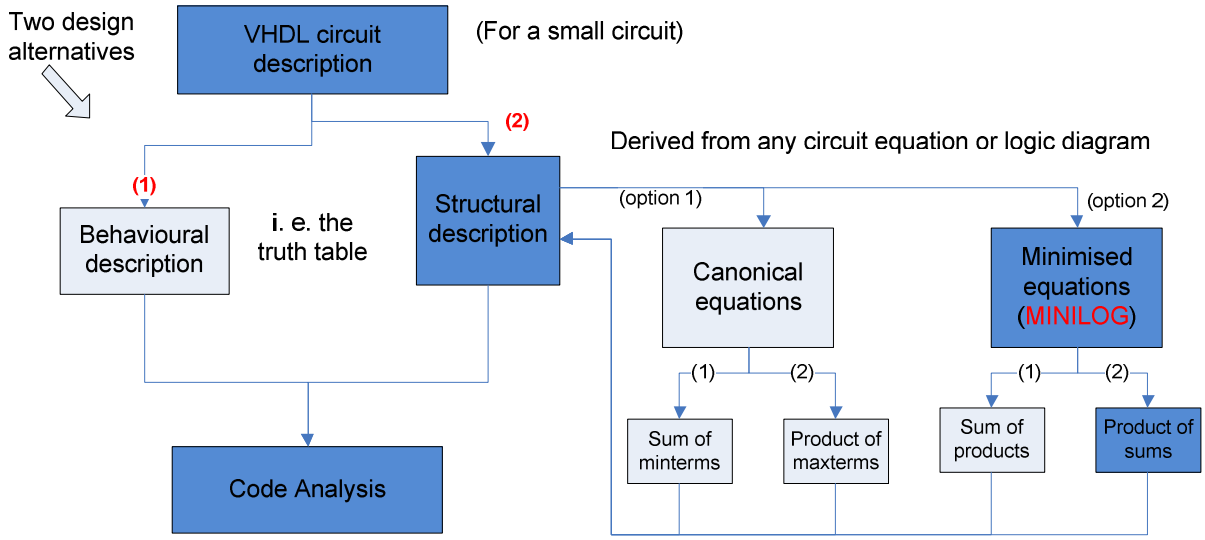


Figura 19 Descripción estructural

Haciendo el código de esta manera controlamos directamente el circuito, es decir, nosotros decidimos cuantos elementos tendrá nuestro circuito y de qué manera se conectan entre ellos.

Una opción sería introducir la suma de Maxterm o el producto de Maxterms en VHDL, pero es poco recomendable ya que no es óptimo.

La otra opción es usar Minilog. Este programa simplifica una tabla de la verdad tanto en una función lógica como en otra tabla de la verdad optimizada. Esto lo hace en suma de productos o en producto de sumas en función de lo que se elija al minimizar la tabla. En la web de la asignatura se puede aprender a usar el Minilog.

Cuando tenemos las funciones lógicas simplificadas ya podemos hacer el código en VHDL.

```
library IEEE;
use    IEEE.STD_LOGIC_1164.all;
entity encoder_8_to_3 is

               port (
                      I7,I6,I5,I4,I3,I2,I1,I0,EI: in STD_LOGIC;
                      EO,GS,S0,S1,S2: out STD_LOGIC
                      );
               end encoder_8_to_3;

architecture logic_equations of encoder_8_to_3 is
               signal V_ENT: STD_LOGIC_VECTOR(8 downto 0);
               signal V_SAL: STD_LOGIC_VECTOR(4 downto 0);

begin
               V_ENT <= EI & I7 & I6 & I5 & I4 & I3 & I2 & I1 & I0;
               S2 <= V_SAL(4);
               S1 <= V_SAL(3);
               S0 <= V_SAL(2);
               GS <= V_SAL(1);
               EO <= V_SAL(0);

               V_SAL(4) <= (V_ENT(8) AND V_ENT(4)) OR (V_ENT(8) AND NOT(V_ENT(6)) AND
V_ENT(5)) OR (V_ENT(8) AND V_ENT(6)) OR (V_ENT(8) AND V_ENT(7));
               V_SAL(3)   <=   (V_ENT(8)   AND   NOT(V_ENT(6))   AND   NOT(V_ENT(5))   AND
NOT(V_ENT(4))  AND  V_ENT(3))  OR  (V_ENT(8)  AND  NOT(V_ENT(5))  AND  NOT(V_ENT(4))  AND
V_ENT(2)) OR (V_ENT(8) AND V_ENT(6)) OR (V_ENT(8) AND V_ENT(7));
               V_SAL(2)   <=   (V_ENT(8)   AND   NOT(V_ENT(6))   AND   NOT(V_ENT(5))   AND
NOT(V_ENT(4))  AND  V_ENT(3))  OR  (V_ENT(8)  AND  NOT(V_ENT(6))  AND  NOT(V_ENT(4))  AND
NOT(V_ENT(2))  AND V_ENT(1)) OR (V_ENT(8) AND NOT(V_ENT(6)) AND V_ENT(5)) OR (V_ENT(8)
AND V_ENT(7));
               V_SAL(1)   <=   (V_ENT(8)   AND   NOT(V_ENT(7))   AND   NOT(V_ENT(6))   AND
NOT(V_ENT(5)) AND NOT(V_ENT(4)) AND NOT(V_ENT(3)) AND NOT(V_ENT(2)) AND NOT(V_ENT(1))
AND NOT(V_ENT(0)));
               V_SAL(0)   <=   (V_ENT(8)   AND   NOT(V_ENT(6))   AND   NOT(V_ENT(5))   AND
NOT(V_ENT(4)) AND V_ENT(3)) OR (V_ENT(8) AND V_ENT(0)) OR(V_ENT(8) AND NOT(V_ENT(6)) AND
NOT(V_ENT(4))  AND  NOT(V_ENT(2))  AND  V_ENT(1))  OR  (V_ENT(8)  AND  NOT(V_ENT(5))  AND
NOT(V_ENT(4))  AND V_ENT(2)) OR (V_ENT(8) AND V_ENT(4)) OR (V_ENT(8) AND NOT(V_ENT(6))
AND V_ENT(5)) OR (V_ENT(8) AND V_ENT(6)) OR (V_ENT(8) AND V_ENT(7));

end logic_equations;
```

3.2.3. Comparación entre funcional y estructural

En este apartado veremos las diferencias entre programar de manera funcional y de manera estructural. De ambas maneras el resultado final es el mismo, pero el circuito físico es diferente. De la manera estructural diseñamos el circuito de tal manera que sabemos cómo será de forma física, en cambio, de la manera funcional no podemos saber de que manera será el circuito físico.

Para ver estas diferencias *Xilinx* tiene una herramienta muy apropiada, que es ver el circuito RTL. A continuación vemos los dos circuitos.

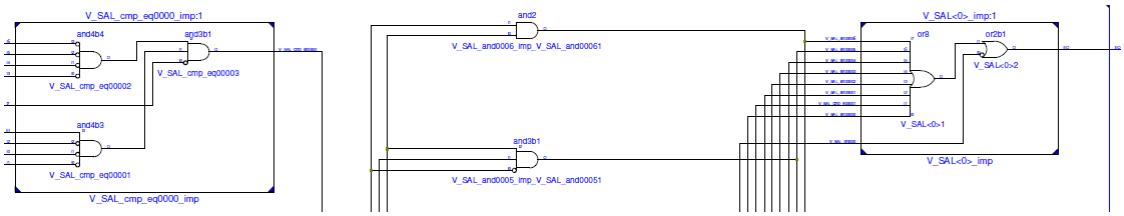


Figura 20 Parte del circuito RTL. Descripción funcional

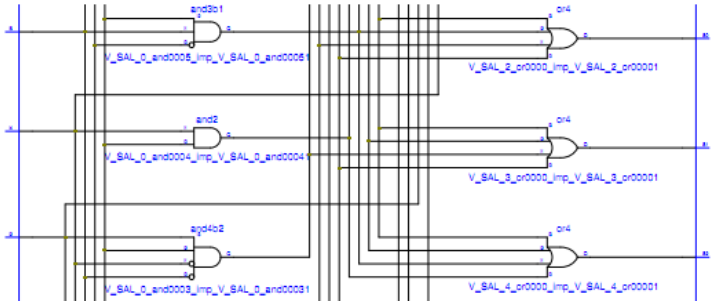


Figura 21 Parte del circuito RTL. Descripción estructural

3.2.4. Simulación

Ahora procederemos a hacer una simulación para comprobar que nuestros circuitos funcionan correctamente. Sólo es necesario hacer un *testbench* ya que realizándolo se puede comprobar si los dos circuitos cumplen con las especificaciones indicadas anteriormente.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY Sim_encoder_8_to_3 IS
END Sim_encoder_8_to_3;

ARCHITECTURE behavior OF Sim_encoder_8_to_3 IS

    COMPONENT encoder_8_to_3
    PORT(
                I7,I6,I5,I4,I3,I2,I1,I0,EI: in STD_LOGIC;
                EO,GS,S0,S1,S2: out STD_LOGIC
        );
    END COMPONENT;

       signal V_ENT: STD_LOGIC_VECTOR(8 downto 0);
       signal V_SAL: STD_LOGIC_VECTOR(4 downto 0);

BEGIN

   uut: encoder_8_to_3 PORT MAP (
          EI => V_ENT(8),I7 => V_ENT(7),I6 => V_ENT(6),I5 => V_ENT(5),I4 => V_ENT(4),I3
=> V_ENT(3),I2 => V_ENT(2),I1 => V_ENT(1),I0 => V_ENT(0),
          EO => V_SAL(4),GS => V_SAL(3),S2 => V_SAL(2),S1 => V_SAL(1),S0 => V_SAL(0)
        );
   stim_proc: process
   begin
               wait for 100 ms;
               V_ENT <= "0--------";
               wait for 100 ms;
               V_ENT <= "100000000";
               wait for 100 ms;
               V_ENT <= "100000001";
               wait for 100 ms;
               V_ENT <= "10000001-";
               wait for 100 ms;
```

```
                V_ENT <= "1000001--";
                wait for 100 ms;
                V_ENT <= "100001---";
                wait for 100 ms;
                V_ENT <= "10001----";
                wait for 100 ms;
                V_ENT <= "1001-----";
                wait for 100 ms;
                V_ENT <= "101------";
                wait for 100 ms;
                V_ENT <= "11-------";
                wait for 100 ms;
      wait;
   end process;

END;
```

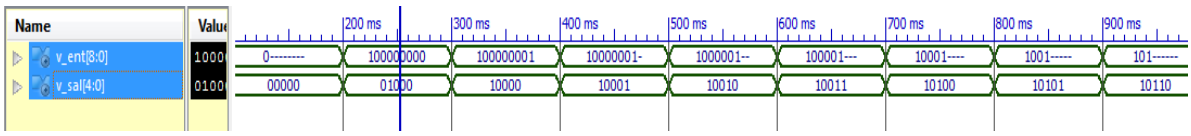


Figura 22 Simulación codificador 8 a 3

Como podemos observar cumple con las especificaciones y ya está listo para ser probado en la placa *Spartan 3E-500*.

3.2.5. Implementación en la placa *Spartan 3E-500*

Ahora ya podemos asignar las entradas y salidas del circuito diseñado a las entradas y salidas físicas de nuestra placa. Este es el código .ucf usado.

```
NET "I0" LOC = G18;
NET "I1" LOC = H18;
NET "I2" LOC = K18;
NET "I3" LOC = K17;
NET "I4" LOC = L14;
NET "I5" LOC = L13;
NET "I6" LOC = N17;
NET "I7" LOC = R17;
NET "EI" LOC = B18;
NET "EO" LOC = F4;
NET "GS" LOC = R4;
NET "S0" LOC = J14;
NET "S1" LOC = J15;
NET "S2" LOC = K15;
```

Hay otra manera de asignar las entradas y salidas sin tener que hacer el código, tenemos que ir a *Tools -> PlanAhead -> I/O Pin Planning*, o bien como se muestra en la siguiente figura en *User Constraints -> I/O Pin PLanning (PlanAhead)*.

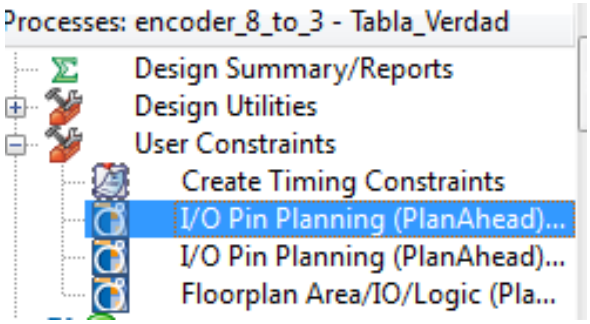


Figura 23 Instrucciones para assignar los pines

Se abre otra ventana y ahí se puede asignar cada entrada y salida una por una.

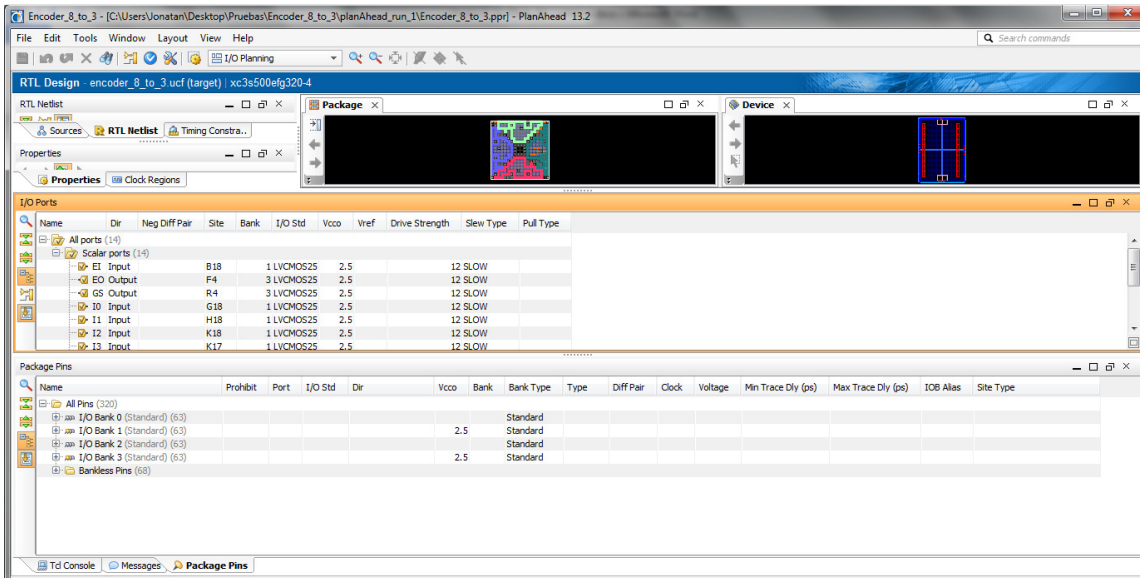


Figura 24 Asignación de pines

Para saber cuáles son los pines que nos interesan tenemos que mirar el ***datasheet*** de nuestra placa.

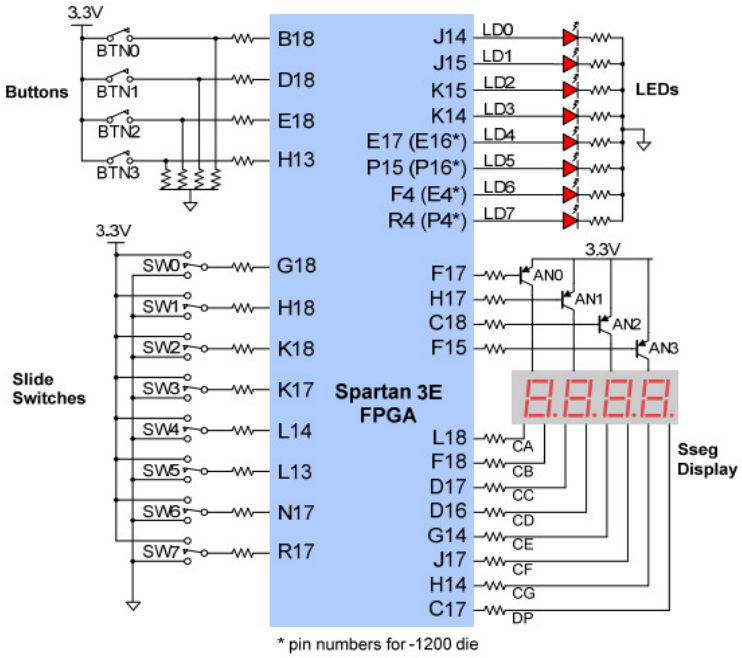


Figura 25 Resumen del *datasheet* de *Spartan 3E-500*

4 CIRCUITOS SECUENCIALES

Los circuitos secuenciales son aquellos cuyas salidas no sólo dependen de las entradas, sino que también dependen del estado anterior o un estado interno. Para comprender un ejemplo de estos circuitos haremos un contador BCD.

También veremos cómo conectar bloques en cascada. Para ello haremos que los números se vean en *Seven Segments.*

4.1. Especificaciones

Este ejercicio constará de 3 entradas y 8 salidas. Una entrada es el reloj interno que tiene *Xilinx,* que es de 50 MHz, otra entrada es el CD, que sirve para resetear el programa y la ultima entrada es el CE, que sirve para encender o apagar el circuito. Siete de las salidas irán conectadas al *Seven Segments* y una irá conectada a un LED que se encenderá intermitentemente cada segundo.

Este proyecto estará dividido en cuatro bloques: un contador BCD, un divisor de frecuencia, un *T-FlipFlop*, y un convertidor *BCD-SevenSegments.*

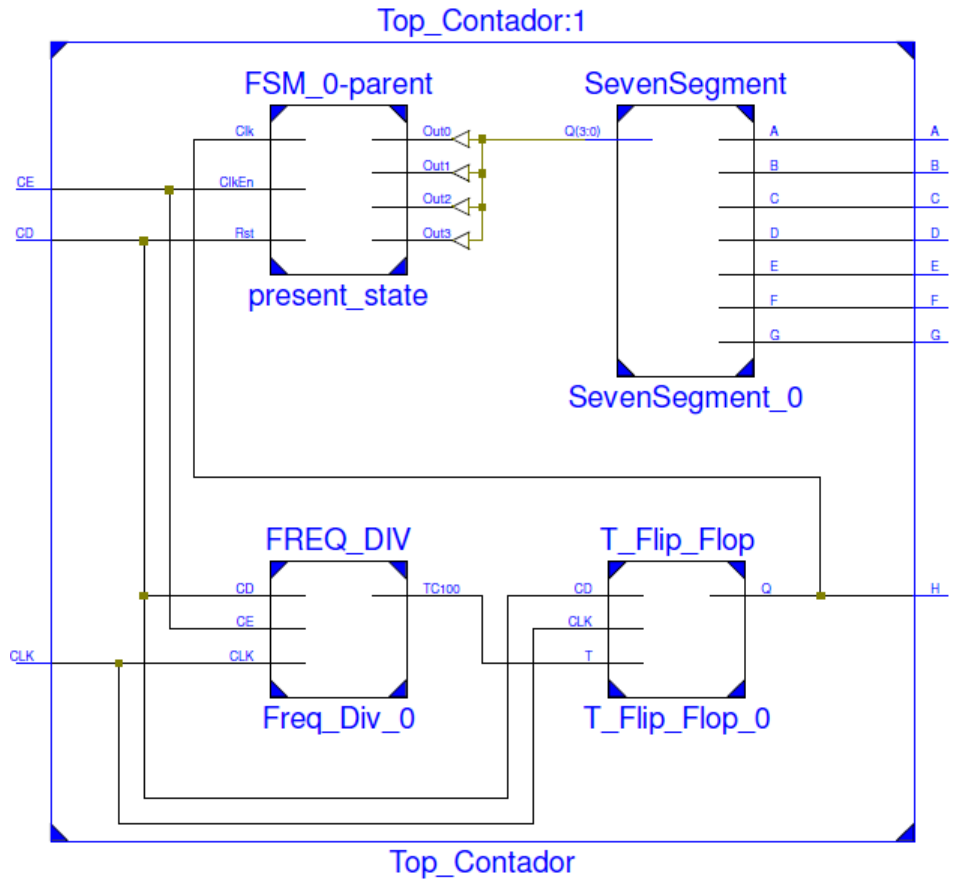


Figura 26 Bloques del contador

4.2. Bloque 1: Contador BCD

Este bloque engloba a todos los demás. La finalidad es muy sencilla, cambia de estado cada vez que le llega una señal de reloj. Y cada estado es un número en BCD.

4.2.1. Esquema y diagrama de estados

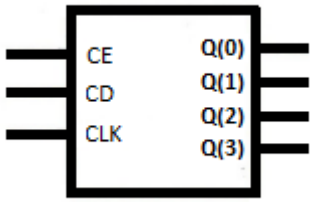


Figura 27 Esquema. Contador

Consta de cuatro salidas para codificar los números del 0 al 9 en BCD. Tiene 3 entradas, una para la señal de reloj, otra para *resetear* el contador y la última para ponerlo en funcionamiento.

Ahora veamos el diagrama de estados.

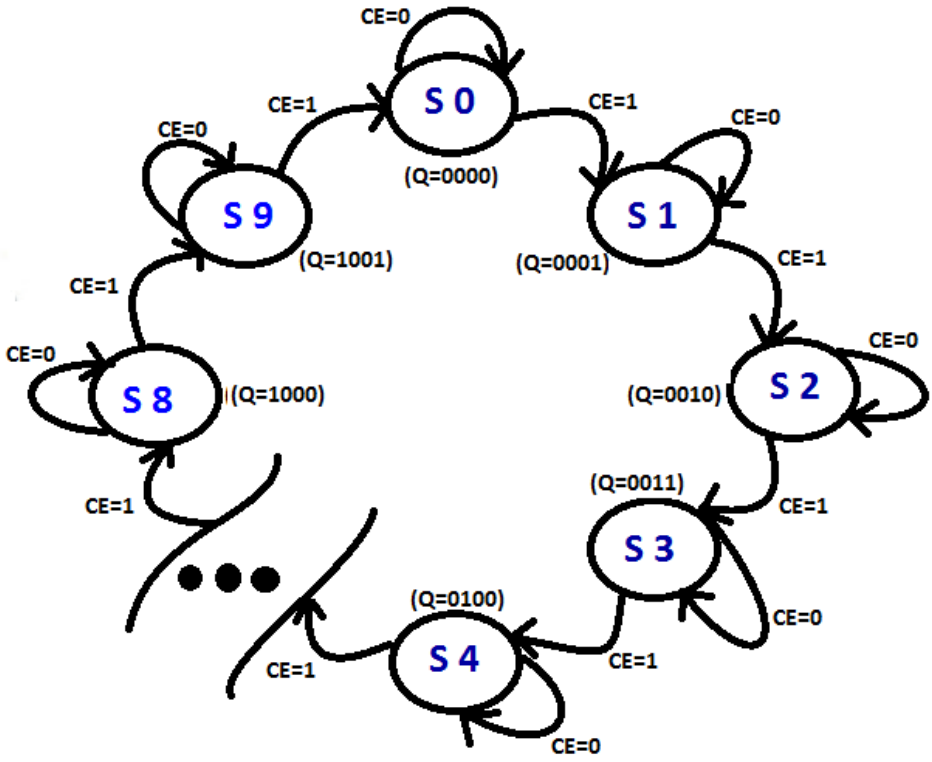


Figura 28 Diagrama de estados. Contador

4.2.2. Diagrama de tiempos

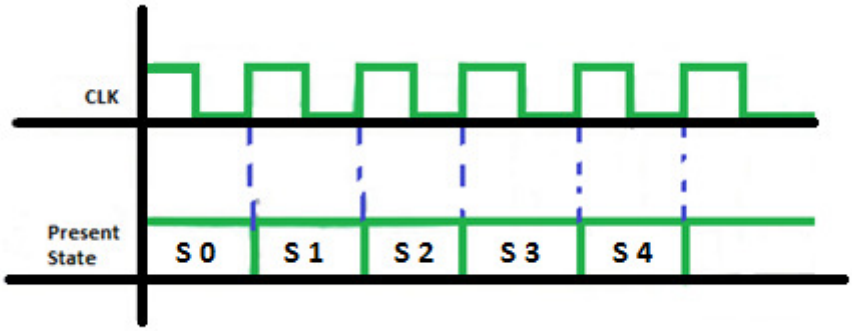


Figura 29 Diagrama de tiempos. Contador

4.2.3. Circuito RTL

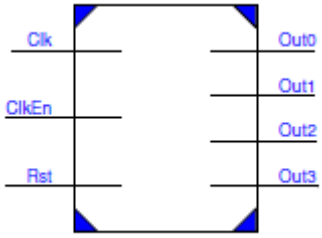


Figura 30 Circuito RTL

4.2.4. Código

El código .vhd se puede encontrar en el *anexo 1.*

4.3. Bloque 2: Divisor de frecuencia

En este proyecto queremos conseguir una onda cuadrada de 1 Hz que será la que determina cada cuándo se cambia de estado; es por eso que tenemos que trabajar con una señal de reloj. La placa *Spartan 3E-500* tiene la posibilidad de añadirle un periférico que puede ser una señal de reloj de la frecuencia deseada, pero también consta de una señal interna de 50 MHz, que es con la que trabajaremos. De ahí la necesidad de este bloque, un divisor de frecuencia.

Tenemos que dividir la frecuencia entre 25000000, y obtendremos una frecuencia de 2 Hz (el próximo bloque se encarga de hacerla cuadrada y dividirla entre 2).

Para realizar este bloque lo explicaremos paso a paso.

4.3.1 Esquema y diagrama de estados

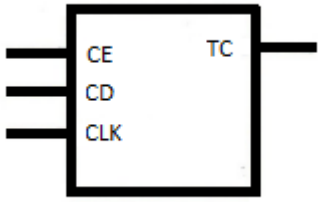


Figura 31 Esquema. Divisor de frecuencia

Como podemos observar este bloque consta de tres entradas y de una salida.

En el diagrama de estados podemos ver como se pasa de un estado a otro, vemos que con el CE=0 se queda continuamente en el mismo estado, mientras que con el CE=1 el estado va cambiando de uno a otro de manera ascendente.

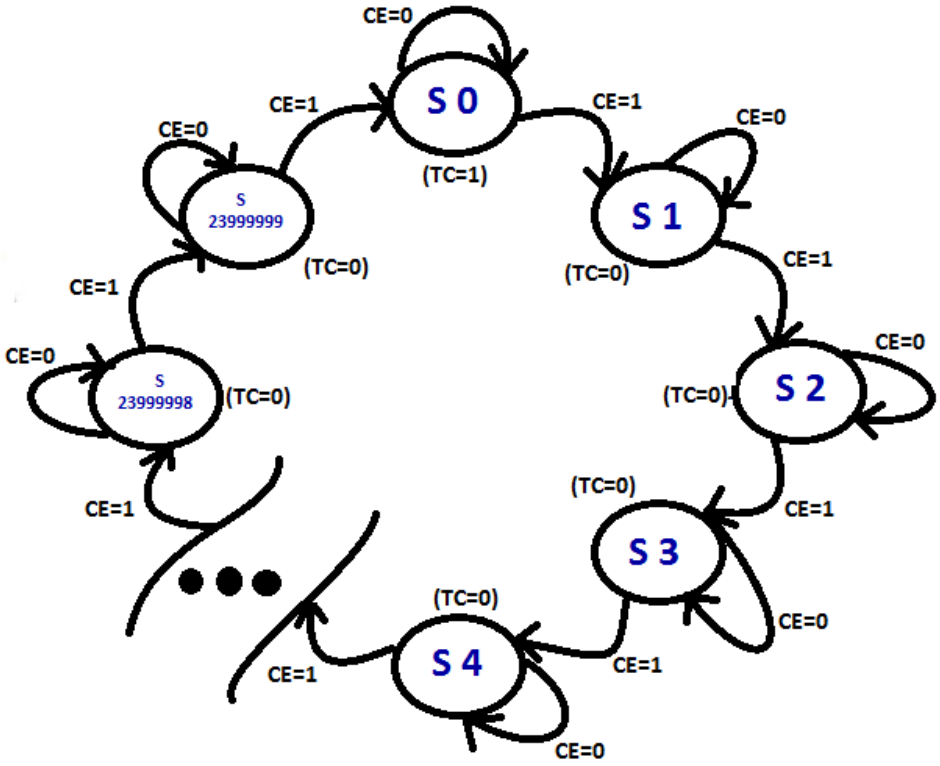


Figura 32 Diagrama de estados. Divisor de frecuencias

4.3.2 Diagrama de tiempos y tabla de estados

Con el diagrama de tiempos podemos ver que con el CD=1 el estado que tenemos es un cero, ya que es un reset.

También podemos observar que cuando CE=0 el sistema se queda en el mismo estado y cuando CE=1 y CD=0 va cambiando de estado.

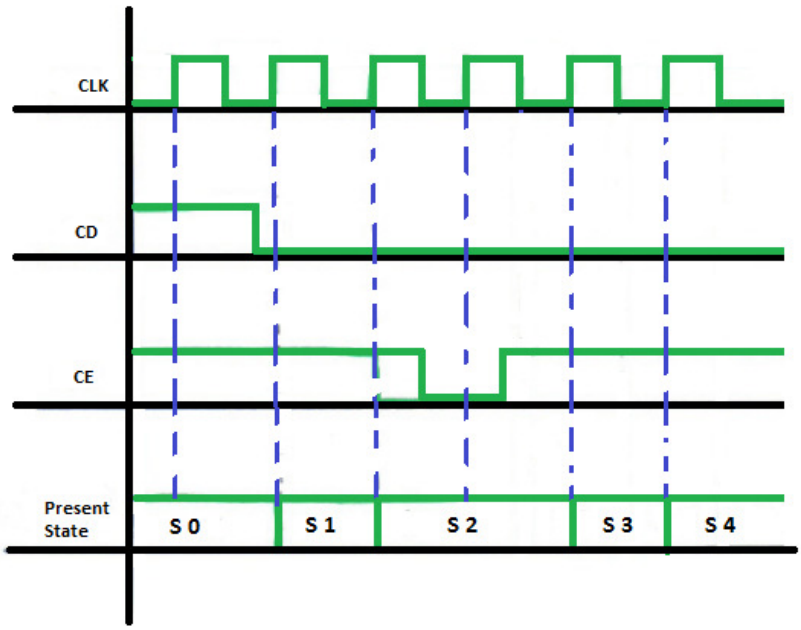


Figura 33 Diagrama de tiempos 1. Divisor de frecuencia

Como tenemos 24000000 estados no podemos ver el diagrama de tiempos entero ya que no apreciaríamos bien el cambio en la salida del TC. Para verlo bien hemos de mirar en la siguiente figura, donde podemos ver que el TC se activa cada 24000000 periodos de CLK.

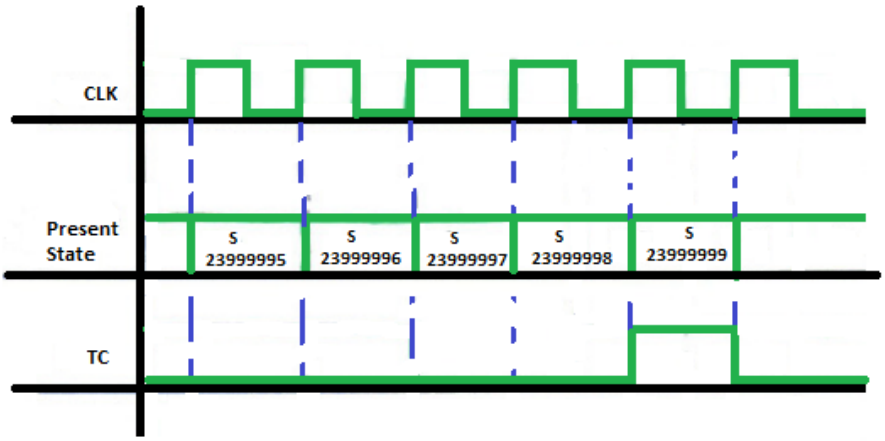


Figura 34 Diagrama de tiempos 2. Divisor de frecuencia

En la siguiente tabla podemos ver como se comporta el circuito en función de su estado anterior.

E	Q+
0 1	Q Q+1

Tabla 2 Tabla de estados. Divisor de frecuencia

Se trata de un biestable del tipo D: cuando está activo el estado actual cambia al siguiente, mientras que cuando se encuentra inactivo el estado actual siempre es el mismo.

4.3.3 Diseño de la maquina de estados finitos *(FSM Design)*

El diseño es el siguiente:

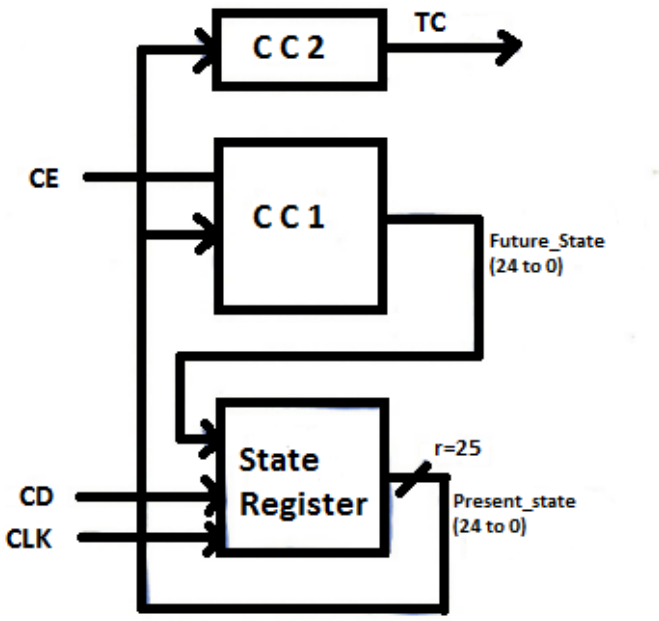


Figura 35 *FSM Design. Divisor de frecuencia*

4.3.4 Circuito RTL

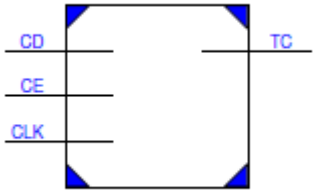


Figura 36 Circuito RTL

4.3.5 Código y simulación

El código del circuito y de la simulación se puede ver en el *Anexo 1.* A continuación podemos observar las gráficas correspondientes a la simulación, las cuales cumplen con las especificaciones iniciales.

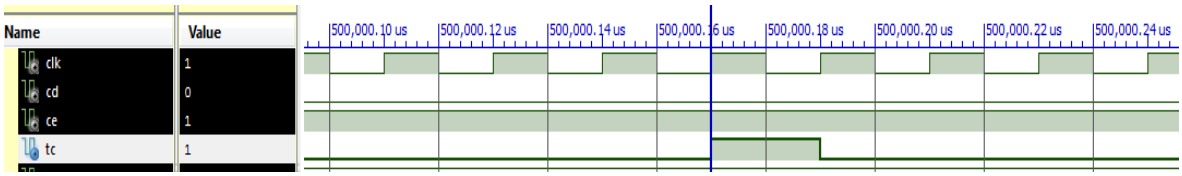


Figura 37 Simulación en Xilinx 2. Divisor de frecuencia

4.4. Bloque 2: *T-Flip Flop*

Nuestra finalidad es conseguir una señal cuadrada de un Hz, por ahora tenemos una señal de 2 Hz. Con un biestable tipo T *(T-Flip Flop)* conseguiremos nuestro objetivo.

Un *T-Flip Flop* no es más que un *D-Flip Flop* pero con una puerta *x-Or* delante, tal y como podemos ver en el siguiente esquema sacado de la web de la asignatura.

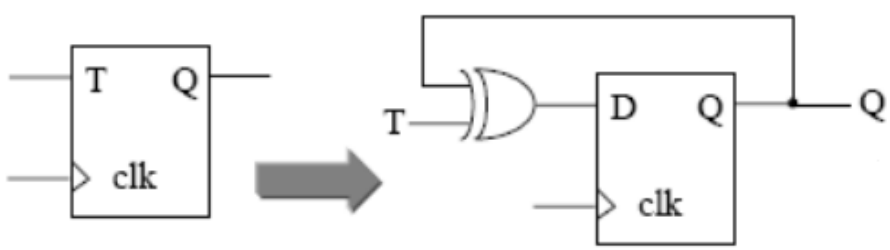


Figura 38 Esquema de un biestable tipo T

4.4.1 Diagrama de tiempos y tabla de estados

A continuación podemos ver como se tiene que comportar este bloque, observando cómo en la salida tenemos una señal cuadrada con la mitad de frecuencia que la de la entrada.

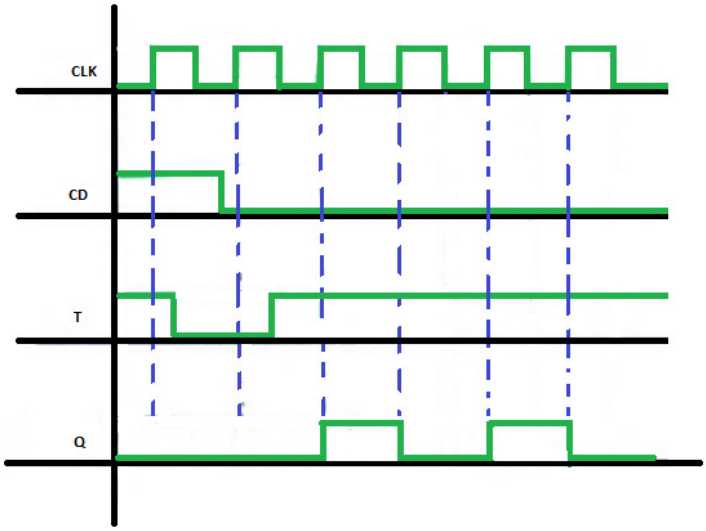


Figura 39 Diagrama de tiempos. *T-Flip Flop*

Tal y como hemos observado anteriormente un *T-Flip Flop* no es más que un *D-Flip Flop* con una puerta *X-Or* delante, por lo tanto su tabla de estados no puede ser otra que la que se muestra a continuación:

T	Q^+
0	Q
1	Q^*

Tabla 3 Tabla de estados. *T-Flip Flop*

4.4.2 Diseño de la maquina de estados finitos (FSM Design)

El diseño es el siguiente:

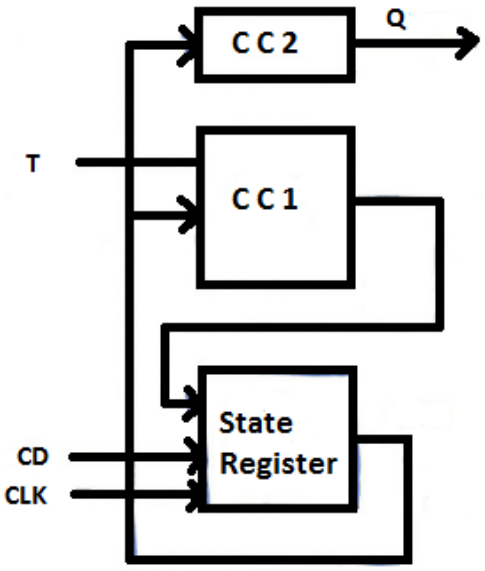


Figura 40 *FSM Design*

4.4.3 Circuito RTL

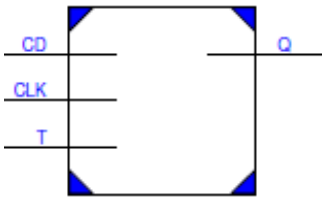


Figura 41 Circuito RTL

4.4.4 Código y simulación

El código del circuito y de la simulación se puede ver en el *Anexo 1.* A continuación podemos observar las gráficas correspondientes a la simulación, las cuales cumplen con las especificaciones iniciales.

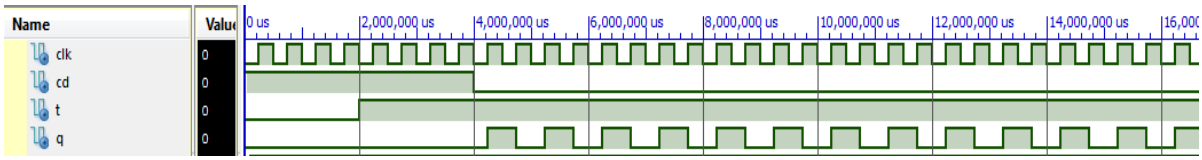


Figura 42 Simulación. *T-Flip Flop*

4.5. Bloque 3: *Seven Segments*

Este bloque sólo se encarga de convertir el código BCD en *Seven Segments.* Esto lo hacemos de manera funcional, es decir, a cada uno de los diez números BCD que tenemos le asignamos que leds se tienen que encender de los displays. El código lo podemos encontrar en el anexo 1.

El circuito RTL obtenido es el siguiente:

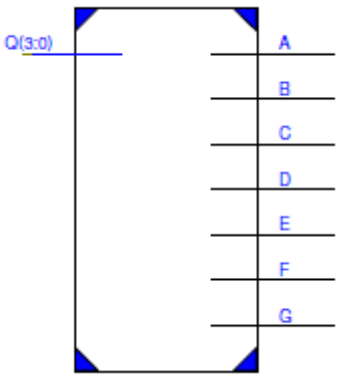


Figura 43 Circuito RTL

5. DISEÑO DE UN UART (*UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER)*

En este capitulo explicaremos el diseño y construcción de un *UART (Universal Asynchronous Receiver-Transmitter)*. Un *UART* permite comunicar una *FPGA* con el PC y viceversa a partir de la norma *RS-232*.

La comunicación desde el PC se hace a través del puerto serie. El ordenador es el *DTE (Data Terminal Equipment)* y la *FPGA* es el *DCE (Data Comunications Equipment)*

Para realizar esta comunicación solo son imprescindibles 3 cables, uno para la transmisión, otro para la recepción y el último para la masa.

Para tener más claro cual es la señal de transmisión y cual es la de recepción tenemos que fijarnos en la siguiente figura, ya que por ejemplo la señal de transmisión de la *FPGA* es la de recepción del *PC*.

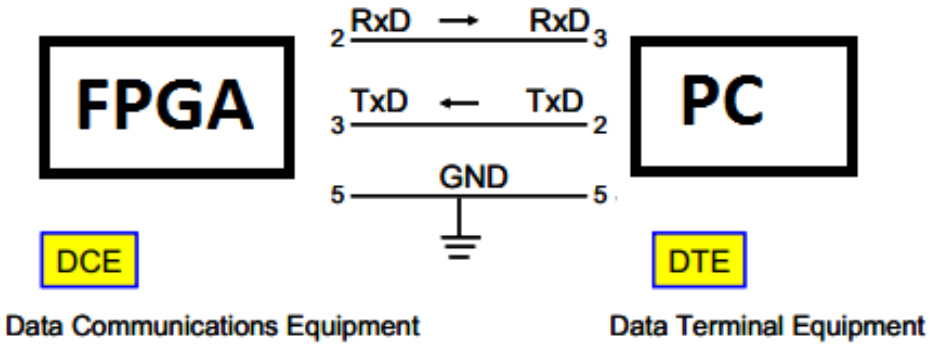


Figura 44 Esquema de la conexión RS-232

A la hora de tener la comunicación entre la FPGA y el PC seguiremos el siguiente protocolo de comunicación RS-232.

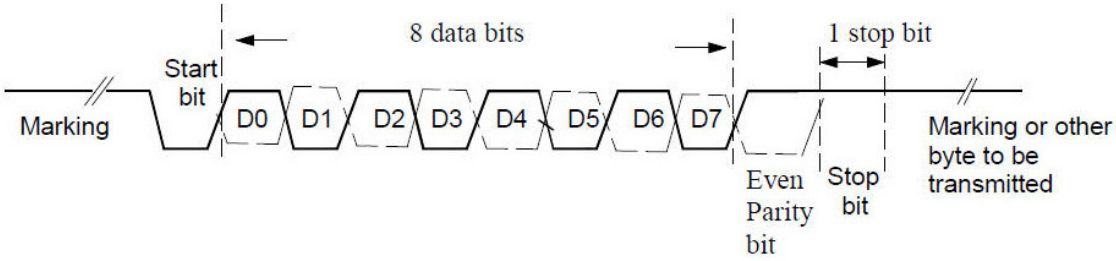


Figura 45 Protocolo de comunicación RS-232

Para la realización de este complejo proyecto tenemos que utilizar numerosos bloques que están estructurados de la siguiente manera.

<table>
<tr><th>Top</th><th>Nivel 1</th><th>Nivel 2</th><th>Nivel 3</th></tr>
<tr><td rowspan="15">UART_module.vhd</td><td rowspan="4">Chip1: Baud_Rate_Generator.vhd</td><td>Chip 1: Freq_Div_326.vhd</td><td>Chip 5: T_flip_flop.vhd</td></tr>
<tr><td>Chip 2: Freq_Div_8.vhd</td><td>Chip 6: T_flip_flop.vhd</td></tr>
<tr><td>Chip 3: Freq_Div_96.vhd</td><td>Chip 7: T_flip_flop.vhd</td></tr>
<tr><td>Chip 4: Freq_Div_100.vhd</td><td>Chip 8: T_flip_flop.vhd</td></tr>
<tr><td rowspan="6">Chip2: Transmitter_unit.vhd</td><td rowspan="5">Chip1: Tranmitter_datapath.vhd</td><td>Chip1: Data_register_8bits.vhd</td></tr>
<tr><td>Chip2: Shift_register_10bit.vhd</td></tr>
<tr><td>Chip3: Mux4.vhd</td></tr>
<tr><td>Chip4: Parity_generator_8bit.vhd</td></tr>
<tr><td>Chip5: Counter_mod8.vhd</td></tr>
<tr><td>Chip2: Transmitter_control_unit.vhd</td><td></td></tr>
<tr><td rowspan="7">Chip3: Receiver_unit.vhd</td><td rowspan="6">Chip1: Receiver_datapath.vhd</td><td>Chip1: data_register_8bits.vhd</td></tr>
<tr><td>Chip2: shift_register_10bit.vhd</td></tr>
<tr><td>Chip3: parity_checker_9bit.vhd</td></tr>
<tr><td>Chip4: counter_mod4.vhd</td></tr>
<tr><td>Chip5: counter_mod8.vhd</td></tr>
<tr><td>Chip6: counter_mod10.vhd</td></tr>
<tr><td>Chip2: Receiver_control_unit.vhd</td><td></td></tr>
<tr><td>Chip4: Debouncing_filter.vhd</td><td></td><td></td></tr>
</table>

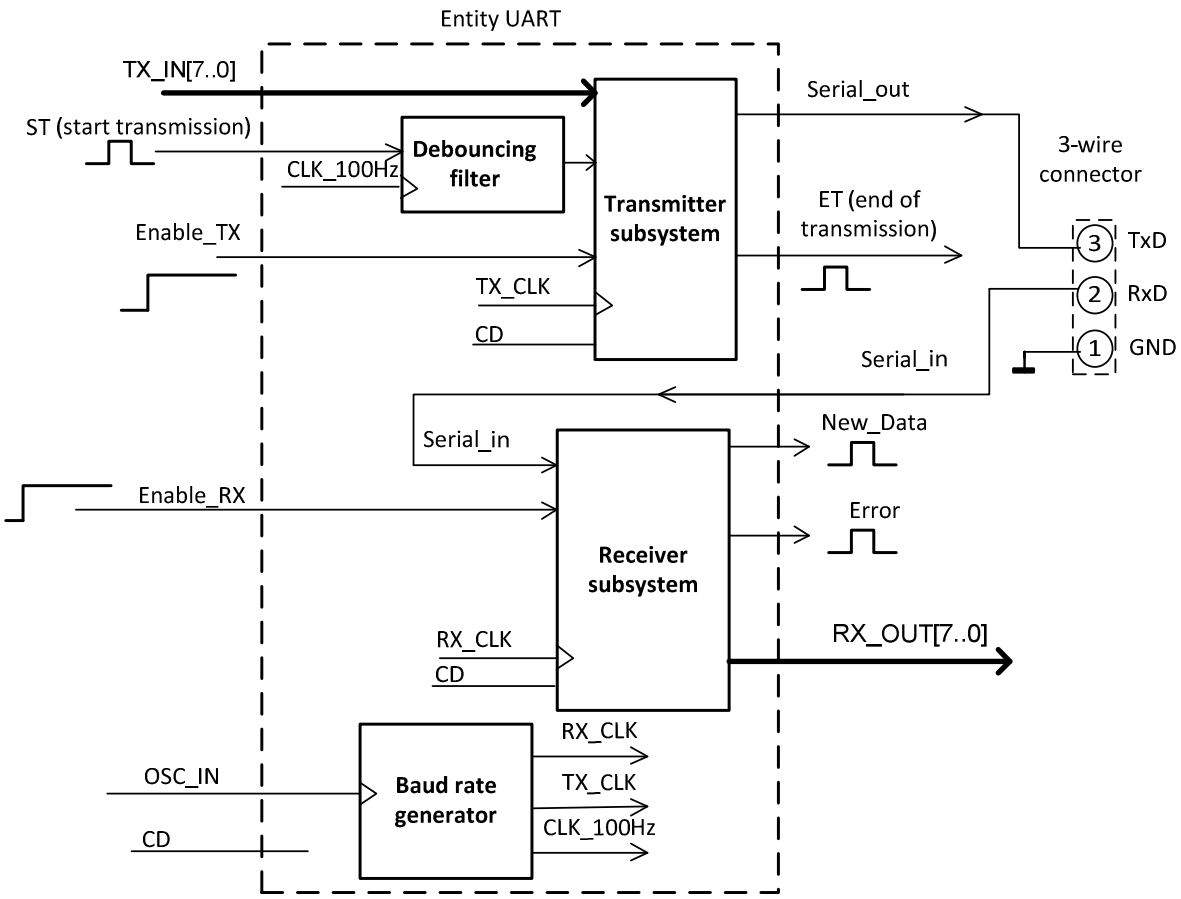


Figura 46 Especificaciones del *UART*

Podemos observar que el *UART* lo forman 4 grandes bloques. Para una mejor comprensión de los mismos, a continuación veremos cada uno de ellos por separado.

5.1 Baud Rate Generator

En esta parte del proyecto haremos varias divisiones de frecuencia. Partiremos de la frecuencia del reloj interno de nuestra placa de Xilinx, que son 50MHz. Para ello utilizaremos la arquitectura FSM y conectaremos varios bloques con dicha arquitectura. Comprobaremos nuestras especificaciones haciendo *testbench* con el simulador que lleva integrado Xilinx en su versión 13.2, que es con la que trabajamos.

5.1.1 Especificaciones

Como podemos observar este proyecto consta de cuatro bloques, el primer bloque dividirá la frecuencia por n=326, el segundo por 8, el tercero por 96 y el cuarto por 100. Con esto conseguimos que a la salida CLK_1Hz_Squared tengamos una señal cuadrada de un Hz.

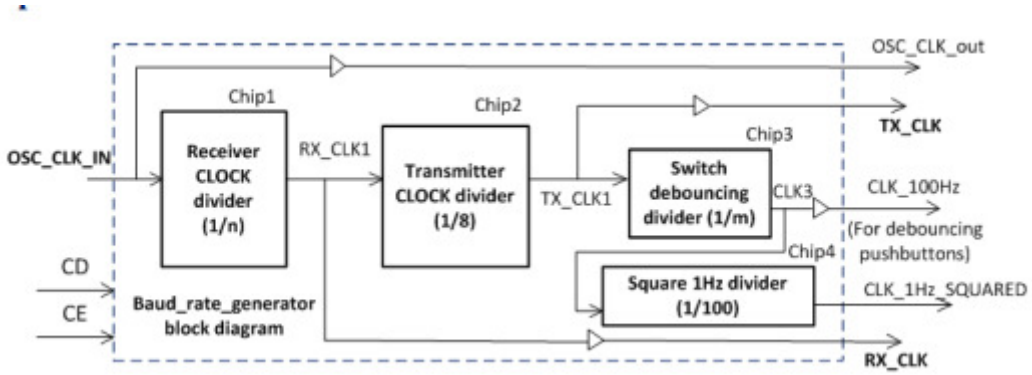


Figura 47 Especificaciones. Baud Rate generator

Ahora estudiaremos cada bloque uno por uno.

5.1.2 Receiver Clock Divider

Este primer bloque divide la frecuencia entre n=651. La manera de hacerlo es la misma que la explicada en el capitulo 4. También tiene dos bloques, uno que divide la frecuencia y el otro que hace la señal cuadrada. La razón por la cual tenemos a la salida una señal cuadrada es empírica, ya que si no lo hacíamos así la señal activa es muy pequeña y a la hora de implementar todo no funciona correctamente.

5.1.3 Transmitter clock divider

Este segundo bloque divide la frecuencia entre 8 de la misma manera que el bloque explicado anteriormente.

5.1.4 Switch debouncing divider

Este bloque es idéntico a los dos anteriores pero divide la frecuencia entre 100.

5.1.5 Square 1Hz Divider

En este bloque dividimos la frecuencia entre 96 y a la salida tenemos una señal cuadrada de 1 Hz,la cual pondremos en un *LED* y nos servirá para saber si el bloque está en funcionamiento y si las divisiones de frecuencia se han hecho correctamente.

5.2 Transmitter Unit

Este bloque está formado por otros dos grandes bloques, que se estructuran de la siguiente manera.

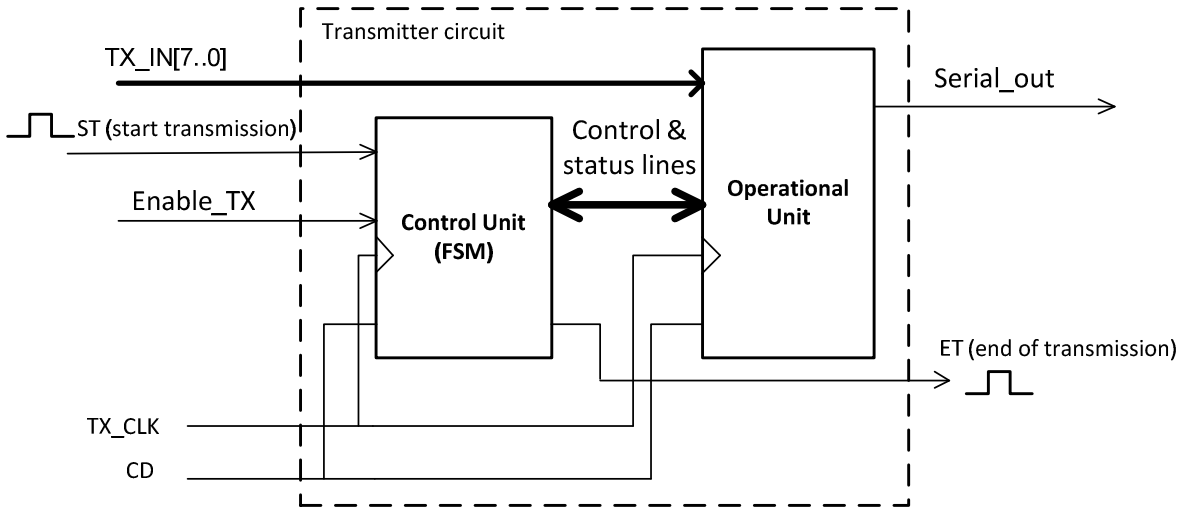


Figura 48 Especificaciones. Transmitter unit

Ahora estudiaremos cada uno de los bloques por separado.

5.2.1 Control Unit (FSM)

Este bloque es el que se encarga de controlar la transmisión. En la siguiente figura podemos ver las especificaciones del mismo.

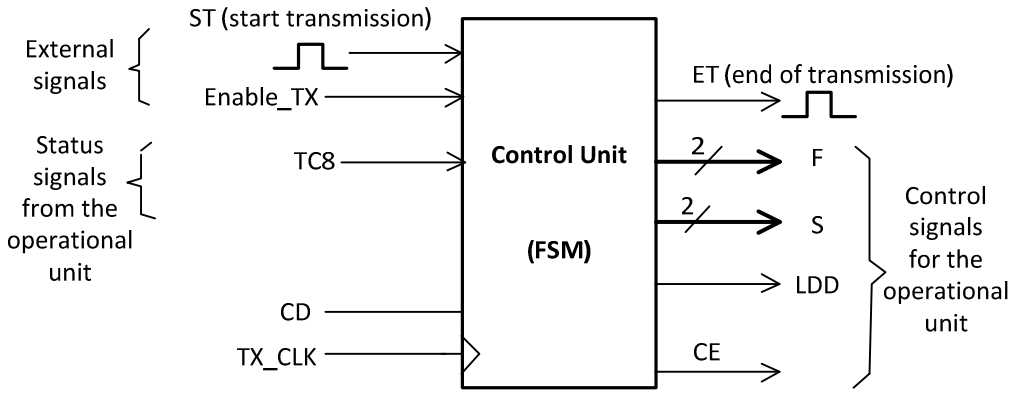


Figura 49 Especificaciones. Control Unit

5.2.2 Datapath Unit

Este bloque está formado por varios circuitos que se estructuran de la siguiente manera.

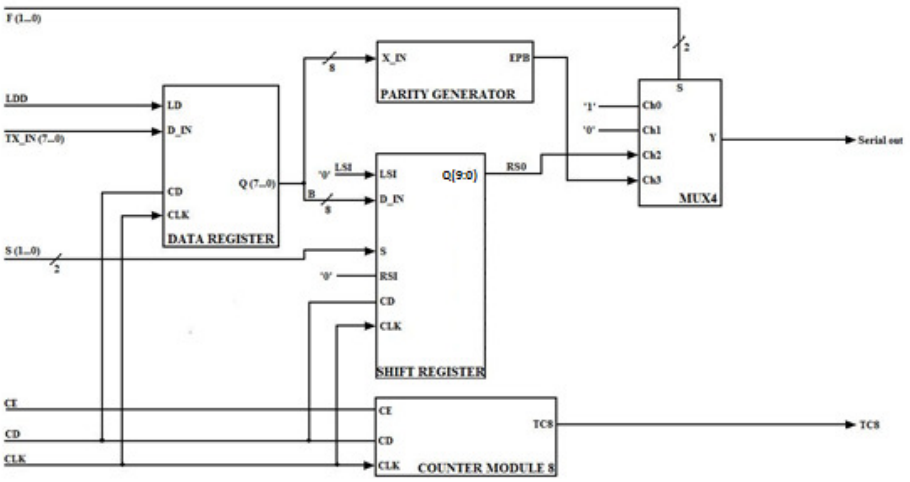


Figura 50 Datapath Unit

Como podemos ver està formado por:

- Data Register (8 bits)
- Shift Register (10 bits)
- Counter module 8
- Multiplexor
- Generador de paridad

Todos estos circuitos están en el anexo 2.

5.3 Receiver Unit

Este bloque es el que se encarga de controlar la recepción. En la siguiente figura podemos sus especificaciones.

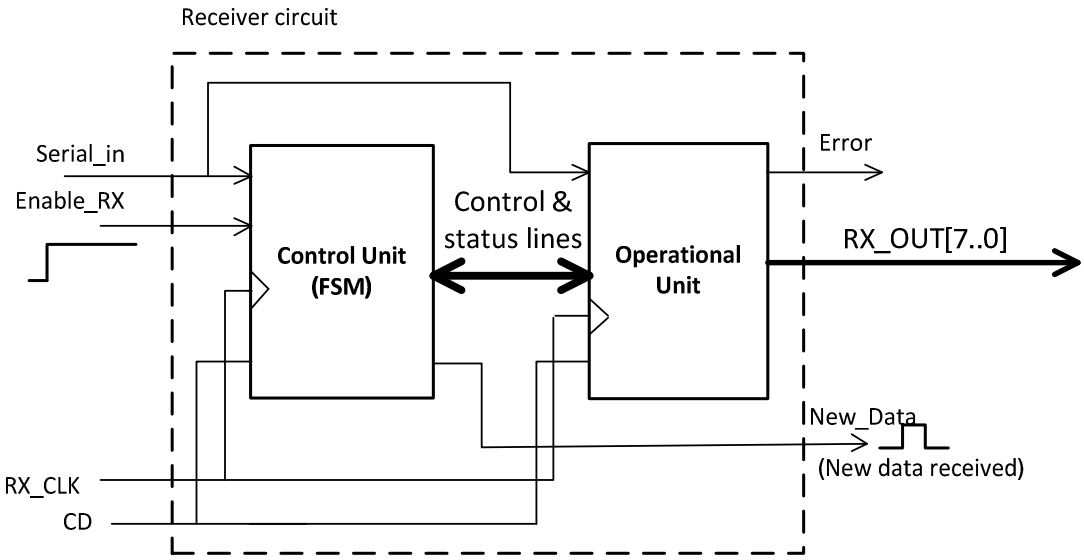


Figura 51 Especificaciones. Circuito receptor

5.3.1 Control Unit (FSM)

Las especificaciones del circuito que se encarga de controlar la recepción de los datos es la siguiente.

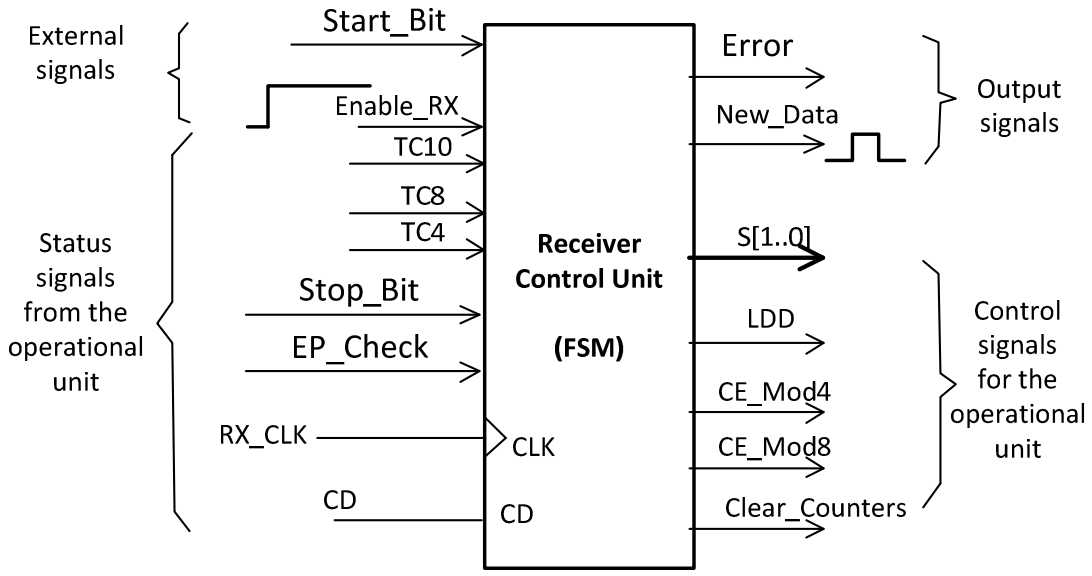


Figura 52 Especificaciones. Control unit

5.3.2 Datapath Unit

Este bloque utiliza los siguientes circuitos que se encuentran en el anexo 2:

- Shift Register (8 bits)
- Data Register (10 bits)
- Control de paridad
- Tres moduladores en cascada

A continuación podemos ver el circuito RTL, donde vemos como se conectan entre sí todos los bloques usados en el datapath.

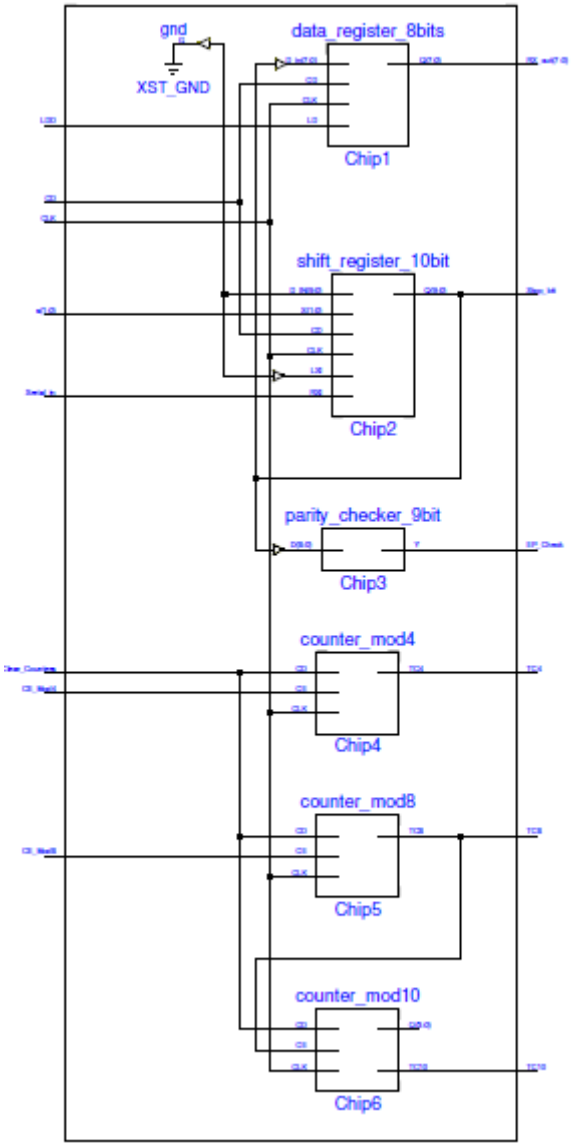


Figura 53 Circuito RTL. Datapath Unit

5.4 Debouncing filter

Este bloque sirve para limpiar la señal de ruido eléctrico que se produce al pulsar un botón. La siguiente figura explica gráficamente de lo que se encarga este último bloque del proyecto.

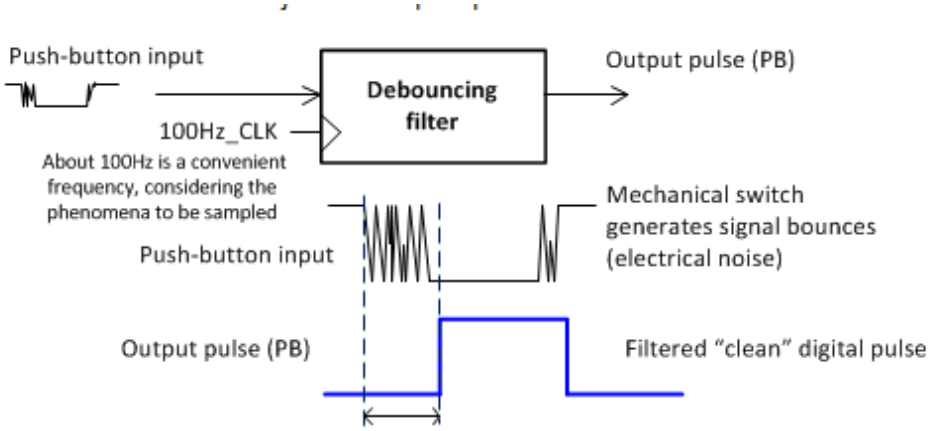


Figura 54 Debouncing filter

El código de este bloque también lo podemos encontrar en el anexo 2.

6 Traffic light controller

Este proyecto lo realizaron los alumnos del curso QT-10/11.

Hemos cogido este proyecto para comprobar visualmente que el UART funciona correctamente.

Haremos un circuito que tenga dos grandes bloques: el *UART* y el *traffic light controller.* Esto lo explicaremos en el capitulo 7.

Además crearemos una interfaz en Labview para ver el correcto funcionamiento de todo el circuito y la comunicación vía puerto serie del PC con la FPGA y viceversa.

Como hemos dicho antes, el proyecto *Traffic light controller* lo hicieron antiguos alumnos de la escuela. En la página web de la asignatura se puede ver el proyecto completo.

A continuación podemos ver el circuito RTL.

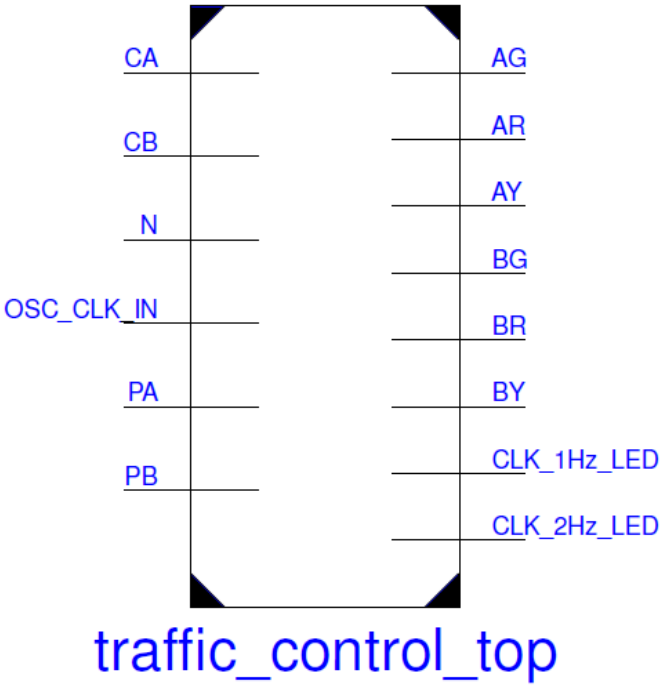


Figura 55 Circuito RTL. Traffic light controller

Como podemos observar consta de 6 entradas y de 8 salidas.

A continuación explicamos brevemente la funcionalidad de cada una de ellas.

- *CA, CB*: con un 0 indican que no hay coches en la calle A/B y con un 1 indican que sí hay coches.

- *N*: con un 1 se activa el modo nocturno, en el cual parpadea *AY* y *BY* simultáneamente.
- *PA, PB*: indican que hay pasajeros en las calles A/B.
- *OSC_CLK*_IN es la señal de reloj, en este caso es el reloj interno de 50 MHz
- *AG, AY, AR:* representan las luces de los semáforos de la calle A.
- *BG, BY, BR*: representan las luces de los semáforos de la calle B.
- *CLK_1Hz_LED, CLK_2Hz_LED*: son señales cuadradas de 1 y 2 Hz.

El proyecto consta de cuatro bloques, con el circuito RTL podemos ver como están interconectados entre ellos.

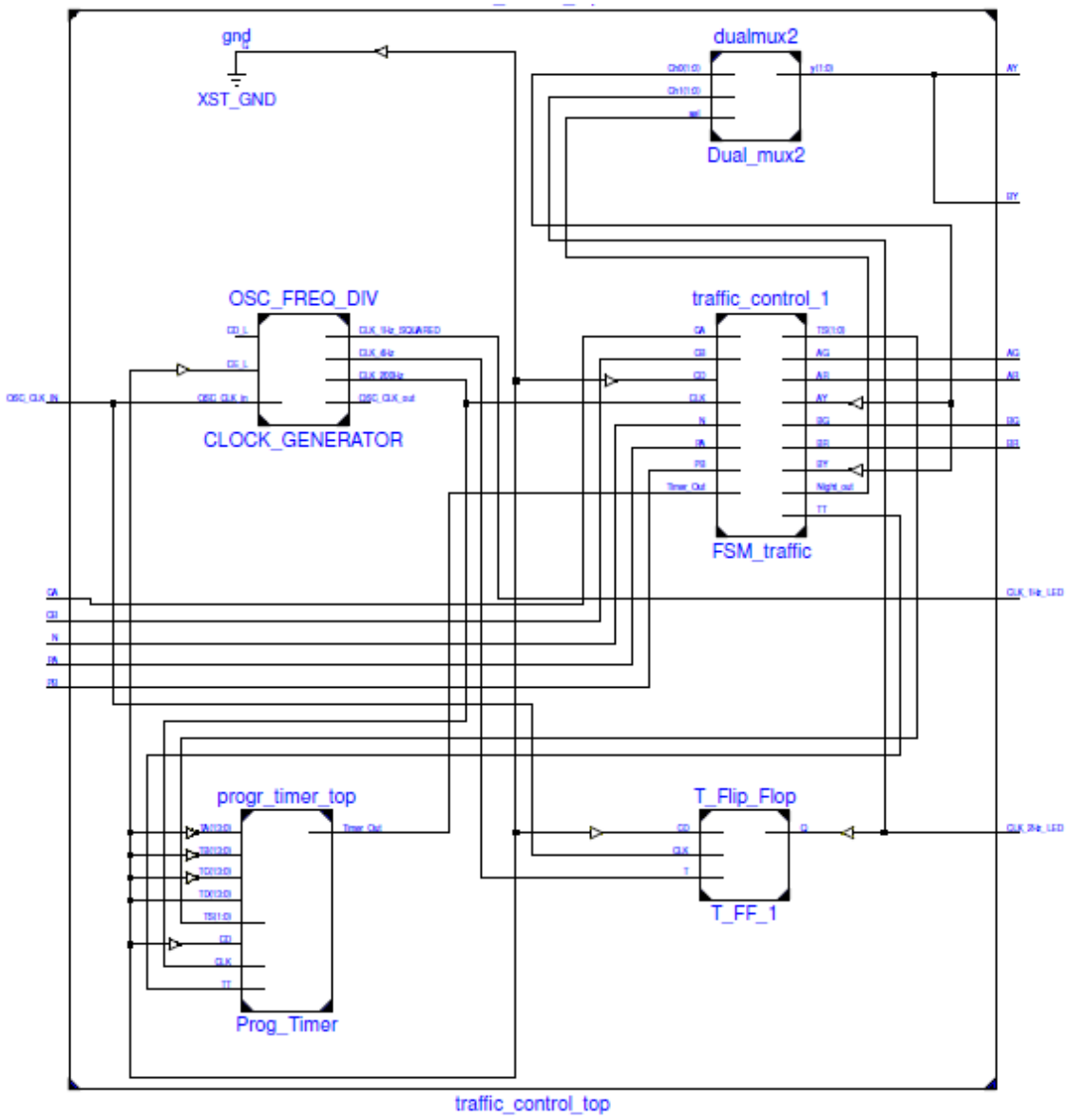


Figura 56 Circuito RTL. Traffic light controller

7 TOP. CIRCUITO GLOBAL

Como hemos dicho antes en este capitulo diseñaremos un circuito que englobe los dos circuitos explicados anteriormente.

El circuito global constará de 5 entradas y 10 salidas.

Las 5 entradas son:

- Señal de reloj, el reloj interno de la placa *Nexys 2* es de 50 MHz.
- *CD (Clear Direct)*, sirve para resetear el circuito. Será un pulsador.
- *Enable Tx*, sirve para habilitar la transmisión de datos. Será un interruptor.
- *Enable Rx*, sirve para habilitar la recepción de datos. Será un interruptor.
- *Serial in,* por este pin entran los datos que provienen del pc.

Las 10 salidas son:

- *Serial out*, es la salida por la cual salen los datos hacia el pc.
- Señal de reloj, es una señal cuadrada de 1Hz. Sirve para saber si el circuito está en funcionamiento y las divisiones de frecuencia están bien hechas.
- 8 salidas simulan las luces de los semáforos. Serán *LEDs* de la placa.

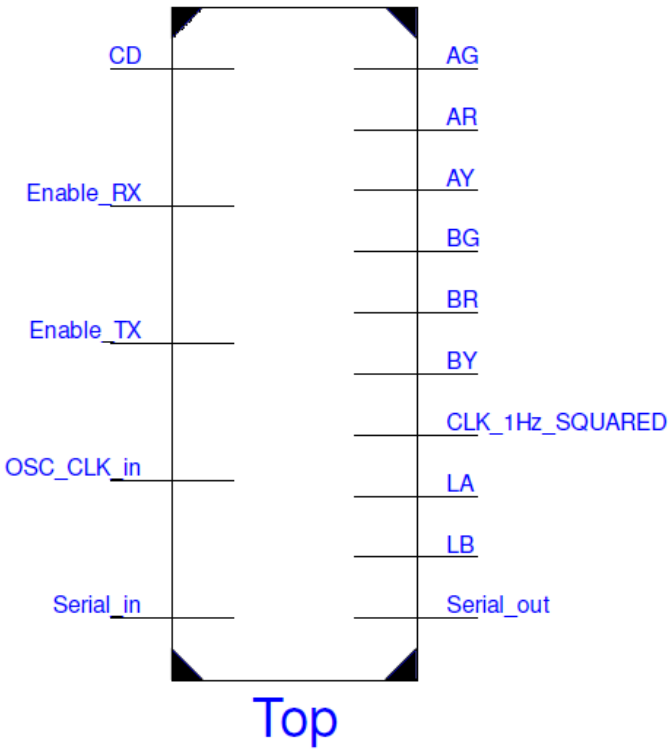


Figura 57 Especificaciones. Bloque global

Las conexiones más importantes entre los dos bloques son las de entradas/salidas del puerto serie. Este es un punto importante ya que entrada/salida depende del punto de vista desde el cual se está mirando. A continuación podemos ver como son estas conexiones entre los dos bloques.

- Salidas del bloque Semáforos -> Entrada al UART
- Salidas del UART -> Entrada al bloque Semáforos
- Entrada de datos del bloque global -> Entrada *Serial_in* del *UART*
    - Salida *Serial_out* del *UART* -> Salida de datos del bloque global

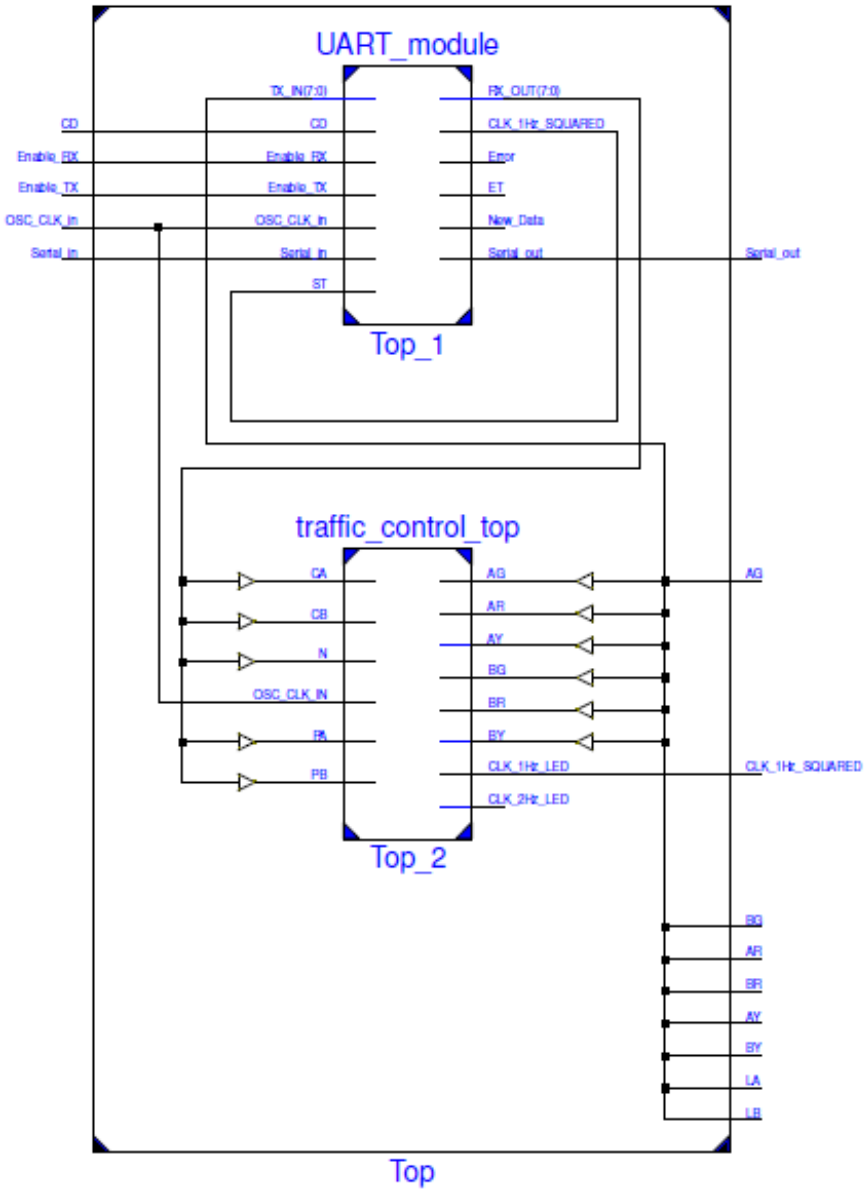


Figura 58 Especificaciones. Interconexión entre los dos bloques

8 INTERFAZ GRÁFICA. LABVIEW

Para ver visualmente la comunicación entre el *PC* y la *FPGA* hemos elegido el *labview*, una herramienta gráfica que permite la comunicación *RS-232* a través del puerto serie.

Este programa funciona de manera gráfica, es decir no hay que insertar código.

El proyecto diseñado permite comunicarse con la FPGA tanto para transmitir como para recibir datos.

A la izquierda tenemos los parámetros que podemos elegir, numero de bits (8), numero de bits de parada (1) y puerto por el que queremos comunicarnos.

En el centro tenemos un cruce de carreteras con todos los semáforos, que podemos ver como funcionan de manera muy visual. En cada carretera tenemos un pulsador que representa si hay coches o no en esa carretera.

En la imagen siguiente podemos verlo.

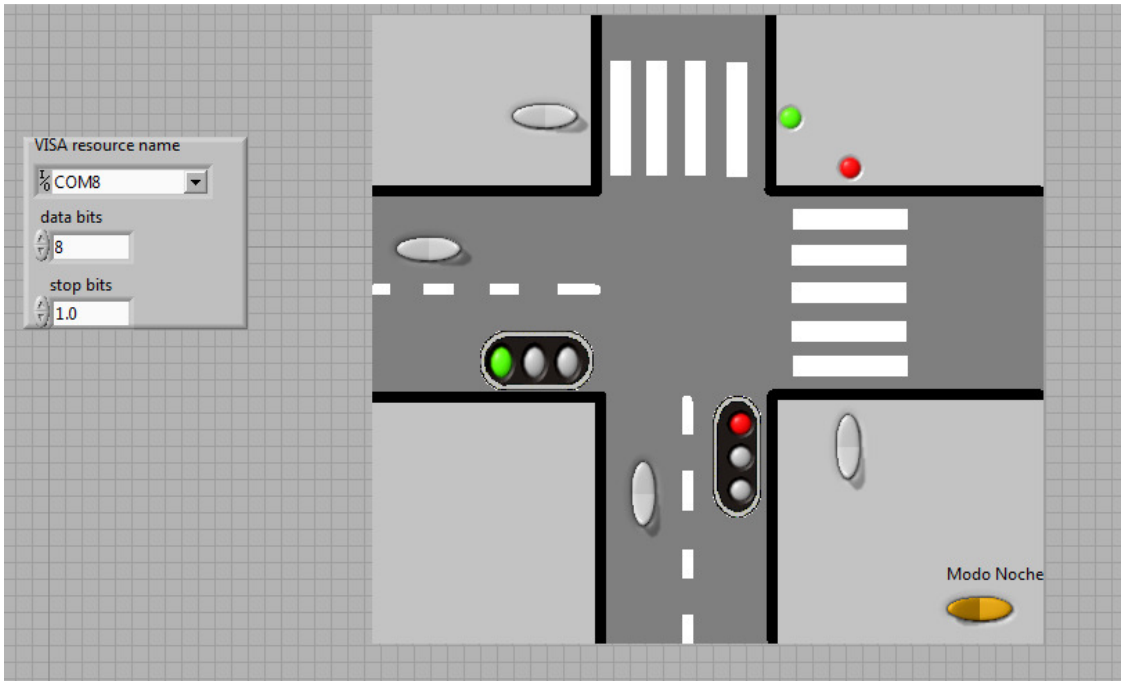


Figura 59 Interfaz gráfica

La programación se hace de forma gráfica, es la siguiente:

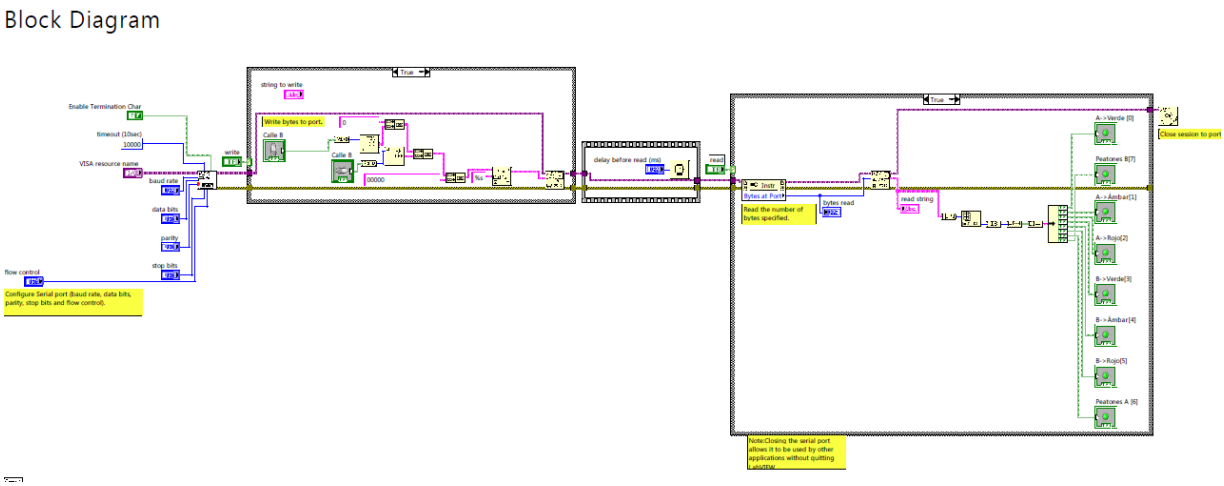


Figura 60 Código interfaz gráfica

9 POSIBLES MEJORAS

La finalidad de este proyecto es hacer pequeños ejercicios a modo de ejemplo para que los futuros alumnos tengan más documentos a la hora de empezar a programar en VHDL. Por lo tanto, las posibles mejoras es hacer muchos más ejercicios con la finalidad de que sea más fácil aprender un lenguaje de programación nuevo.

En cuanto al ejercicio principal, se podría generar una base de datos para saber las estadísticas de cada calle y en función de dicha estadística actuar de una manera u otra.

Sobre el UART se podría mejorar haciendo un buffer donde se guardaran los datos cuando van demasiado rápido. También se podría hacer que en vez leer byte a byte se pudiera escoger el tamaño de los paquetes de datos de entrada y/o salida.

La comunicación RS-232 se podría sustituir por una comunicación vía *bluetooth* o vía *wireless*.

10 CONCLUSIONES

La finalidad de este proyecto era tener un documento en el cual basarse a la hora de empezar a programar estos dispositivos en un lenguaje de programación nuevo.

Ha sido de gran utilidad hacer este proyecto ya que durante la carrera no había tenido la oportunidad de aprender este leguaje y creo que es muy importante aprenderlo si te interesan los circuitos digitales.

Al principio cuesta entender porque se hace así y no con una interfaz más visual como puede ser *Proteus*, pero cuando se hacen proyectos más complejos se da uno cuenta de que la mejor forma de estructurarlos es mediante circuitos descritos en VHDL.

En este proyecto hemos podido ver ejercicios de todo tipo, sencillos para poder empezar a programar y complejos como es la realización de un UART.

Una de las cosas que me ha parecido más positiva a la hora de realizar este proyecto ha sido poder aprender a usar varios programas, como son *ISE Project Navigator, Minilog y Labview.*

ANNEXOS

TÍTULO DEL TFC: Aplicaciones didácticas de PLD/FPGA para las asignaturas de sistemas digitales

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad Sistemas de Telecomunicaciones / Ingeniería Técnica de Aeronáutica, especialidad Aeronavegación

AUTOR: Jonatan González Rodríguez

DIRECTOR: Francesc Josep Sánchez i Robert

Fecha: 24 de marzo de 2012

Anexo 1 Códigos. Circuitos secuenciales

Código .vhd Contador

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

entity Top_Contador is
       Port ( CLK     : IN STD_LOGIC;
                        CD      : IN    STD_LOGIC;
                        CE      : IN    STD_LOGIC;
                        H                               : OUT   STD_LOGIC;
                        A,B,C,D,E,F,G  : out STD_LOGIC
                    );
end Top_Contador;


architecture Contador of Top_Contador is

COMPONENT FREQ_DIV IS

       PORT
              (
               CE,CD,CLK     : IN    STD_LOGIC;
               TC            : OUT   STD_LOGIC
              );
END COMPONENT;

COMPONENT T_Flip_Flop IS

       PORT
              (
               T,CD,CLK      : IN    STD_LOGIC;
               Q              : OUT  STD_LOGIC
              );
END COMPONENT;

COMPONENT SevenSegment IS

       PORT
              (
               Q             : IN    STD_LOGIC_VECTOR(3 DOWNTO  0);
               A,B,C,D,E,F,G : out STD_LOGIC
              );
END COMPONENT;
-- Internal wires
                            -- State signals declaration
                            -- A BCD counter will consist of 10 diferent states
       TYPE State_type IS (Num0, Num1, Num2, Num3, Num4, Num5, Num6, Num7, Num8, Num9) ;
--------> This is important: specifying to the synthesiser the code for the states
       ATTRIBUTE syn_encoding : STRING;
       ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";
              -- (It may be:"sequential" (binary); "gray"; "one-hot", etc.
       SIGNAL present_state, future_state   : State_type ;

-- Constants for special states (the first and the last state)
CONSTANT Reset        : State_type := Num0; -- The fisrt state. This is another name for
the state Num0
CONSTANT Max_count    : State_type := Num9; -- The last state. This is anather way to
name the state Num9

SIGNAL CE0,CLK_NEW : STD_LOGIC;
SIGNAL Q              :STD_LOGIC_VECTOR(3 DOWNTO  0);

begin
state_register: PROCESS (CD,CLK_NEW, future_state)
       BEGIN

       IF CD = '1' THEN                     -- reset counter
              present_state <= Reset;
       ELSIF (CLK_NEW'EVENT and CLK_NEW = '1') THEN--   Synchronous   register   (D-type
flip-flop)
```

```
                present_state <= future_state;
        END IF;

END PROCESS state_register;

        Freq_Div_0 : Freq_Div

        PORT MAP (
                                                -- from component name        =>    to
signal or port name
                        CE              => CE,
                        CD              => CD,
                        CLK             => CLK,
                        TC              => CE0
                          );

T_Flip_Flop_0 : T_Flip_Flop

        PORT MAP (
                                                -- from component name        =>    to
signal or port name
                        T               => CE0,
                        CD              => CD,
                        CLK             => CLK,
                        Q               => CLK_NEW
                          );

H <= CLK_NEW;

SevenSegment_0 : SevenSegment

        PORT MAP (
                                                -- from component name        =>    to
signal or port name
                        Q               => Q,
                        A               => A,
                        B               => B,
                        C               => C,
                        D               => D,
                        E               => E,
                        F               => F,
                        G               => G
                          );
------------------------- CC1: Combinational system for calculating next state
CC_1: PROCESS (present_state, CE)
        BEGIN
                IF CE = '0' THEN
                        future_state <= present_state;  -- count disable

                ELSE                                    -- just a simple state up count
                        CASE present_state IS
                                WHEN Num0 =>
                                        future_state <= Num1 ;
                                WHEN Num1 =>
                                        future_state <= Num2 ;
                                WHEN Num2 =>
                                        future_state <= Num3 ;
                                WHEN Num3 =>
                                        future_state <= Num4 ;
                                WHEN Num4 =>
                                        future_state <= Num5 ;
                                WHEN Num5 =>
                                        future_state <= Num6 ;
                                WHEN Num6 =>
                                        future_state <= Num7 ;
                                WHEN Num7 =>
                                        future_state <= Num8 ;
                                WHEN Num8 =>
                                        future_state <= Num9 ;
                                WHEN Num9 =>
                                        future_state <= Num0 ;
                        END CASE ;

                END IF;

END PROCESS CC_1;
```

```
------------------------- CS_2: combinational system for calculating extra outputs
-------------------------- and outputing the present state (the actual count)
CC_2: PROCESS (present_state, CE)
        BEGIN
-- And now just copying the present state to the output:
               CASE present_state IS
                              WHEN Num0 =>
                                     Q <= "0000";
                              WHEN Num1 =>
                                     Q <= "0001";
                              WHEN Num2 =>
                                     Q <= "0010";
                              WHEN Num3 =>
                                     Q <= "0011";
                              WHEN Num4 =>
                                     Q <= "0100";
                              WHEN Num5 =>
                                     Q <= "0101";
                              WHEN Num6 =>
                                     Q <= "0110";
                              WHEN Num7 =>
                                     Q <= "0111";
                              WHEN Num8 =>
                                     Q <= "1000";
                              WHEN Num9 =>
                                     Q <= "1001";
               END CASE ;

END PROCESS CC_2;
-- Place other logic if necessary
end Contador;
```

Código .vhd Divisor de frecuencia

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
ENTITY Freq_Div IS
      Port (   CLK     : IN    STD_LOGIC;
               CD                              : IN    STD_LOGIC;
               CE                              : IN    STD_LOGIC;
               TC                                      : OUT   STD_LOGIC

       );

END Freq_Div;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF Freq_Div IS

CONSTANT Max_Count    : STD_LOGIC_VECTOR(24 DOWNTO 0) := "10111110101111100000111111";

CONSTANT Reset        : STD_LOGIC_VECTOR(24 DOWNTO 0) := "00000000000000000000000000"; --
1100 0000 0001 0001 1110 1100 --1100 0000 0001 0001 1110 1011

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(24 downto 0) := Reset;

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                      -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;
```

```
------------------------ ESS state_registercombinational system for calculating next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
              IF(present_state < Max_Count ) THEN
                     future_state <= present_state + 1 ;
              ELSE
                     future_state <= Reset;
              END IF;
       ELSE
              future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

------------------------ CS_2: combinational system for calculating extra outputs
-------------------------- and outputing the present state (the actual count)

TC <= '1' WHEN ((present_state = Max_Count)AND CE = '1') ELSE '0';  --terminal count

END FSM_like;
```

Simulación *(Test Bench)* .vhd Divisor de frecuencia

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Sim_Freq_Div IS
END Sim_Freq_Div;

ARCHITECTURE behavior OF Sim_Freq_Div IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT Freq_Div
    PORT(
         CLK : IN  std_logic;
         CD : IN  std_logic;
         CE : IN  std_logic;
         TC : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal CLK : std_logic := '0';
   signal CD : std_logic := '0';
   signal CE : std_logic := '0';

       --Outputs
   signal TC : std_logic;

   -- Clock period definitions
   constant CLK_period : time := 20 ns;

BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: Freq_Div PORT MAP (
          CLK => CLK,
          CD => CD,
          CE => CE,
          TC => TC
        );

   -- Clock process definitions
   CLK_process :process
   begin
              CLK <= '0';
              wait for CLK_period/2;
              CLK <= '1';
              wait for CLK_period/2;
   end process;
```

```
   -- Stimulus process
   stim_proc: process
   begin

               CD <= '1';
               CE <= '1';
               wait for 100 ns;
               CD <= '0';
               wait for 100 ns;
               CE <= '0';
               wait for 100 ns;
               CE <= '1';


      -- insert stimulus here

      wait;
   end process;
```

END;Código .vhd *T-Flip Flop*

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY T_Flip_Flop IS
      Port (   CLK     : IN    STD_LOGIC;
               CD      : IN    STD_LOGIC;
               T       : IN    STD_LOGIC;
               Q       : OUT   STD_LOGIC
       );

END T_Flip_Flop;

ARCHITECTURE FSM_like OF T_Flip_Flop IS


CONSTANT Reset : STD_LOGIC := '0';

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC := Reset;
  -- This thing of initialising these signals to the "Reset" state,
  -- is only an issue for the functional simulator. Once the circuit
  -- is synthesised, this thing is completely irrelevant.

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                     -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- next state logic
-- A T flip flop invert the output when T = 1, and do nothing when T = 0

CS_1: PROCESS (present_state, T)
       BEGIN
       IF T = '1' THEN
                     future_state <= NOT (present_state);
       ELSE
              future_state <= present_state;
       END IF;

END PROCESS CS_1;

------------------------ CS_2: combinational system for calculating extra outputs
-- Very simple in this case, a buffer.
Q <= present_state;
```

```
END FSM_like;
```

Simulación *(Test Bench)* .vhd *T-Flip Flop*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Sim_T_Flip_Flop IS
END Sim_T_Flip_Flop;

ARCHITECTURE behavior OF Sim_T_Flip_Flop IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT T_Flip_Flop
    PORT(
         CLK : IN  std_logic;
         CD : IN  std_logic;
         T : IN  std_logic;
         Q : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal CLK : std_logic := '0';
   signal CD : std_logic := '0';
   signal T : std_logic := '0';

       --Outputs
   signal Q : std_logic;

   -- Clock period definitions
   constant CLK_period : time := 500 ms;

BEGIN

       -- Instantiate the Unit Under Test (UUT)
   uut: T_Flip_Flop PORT MAP (
          CLK => CLK,
          CD => CD,
          T => T,
          Q => Q
        );

   -- Clock process definitions
   CLK_process :process
   begin
               CLK <= '0';
               wait for CLK_period/2;
               CLK <= '1';
               wait for CLK_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
      wait for 100 ns;
               CD <='1';
      T <= '0';
               wait for 2000 ms;
               T <= '1';
               wait for 2000 ms;
               CD <= '0';

      -- insert stimulus here

      wait;
   end process;

END;
```

Código .vhd Seven Segments

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SevenSegment is
port (
               Q              : IN   STD_LOGIC_VECTOR(3 DOWNTO  0);
               A,B,C,D,E,F,G  : out STD_LOGIC
       );
end SevenSegment;

architecture Truth_table of SevenSegment is

       signal CA: STD_LOGIC_VECTOR (3 downto 0);
       signal CB: STD_LOGIC_VECTOR (3 downto 0);
       signal CC: STD_LOGIC_VECTOR (3 downto 0);
       signal CD: STD_LOGIC_VECTOR (3 downto 0);
       signal CE: STD_LOGIC_VECTOR (3 downto 0);
       signal CF: STD_LOGIC_VECTOR (3 downto 0);
       signal CG: STD_LOGIC_VECTOR (3 downto 0);

begin

       CA <= Q(3) & Q(2) & Q(1) & Q(0);
       CB <= Q(3) & Q(2) & Q(1) & Q(0);
       CC <= Q(3) & Q(2) & Q(1) & Q(0);
       CD <= Q(3) & Q(2) & Q(1) & Q(0);
       CE <= Q(3) & Q(2) & Q(1) & Q(0);
       CF <= Q(3) & Q(2) & Q(1) & Q(0);
       CG <= Q(3) & Q(2) & Q(1) & Q(0);

       with CA select
              A <= '0' when         "0000" |
                                    "0010" |
                                    "0011" |
                                    "0101" |
                                    "0110" |
                                    "0111" |
                                    "1000" |
                                    "1001",
                           '1' when              others;
       with CB select
              B <= '0' when         "0000" |
                                    "0001" |
                                    "0010" |
                                    "0011" |
                                    "0100" |
                                    "0111" |
                                    "1000" |
                                    "1001",
                           '1' when      others;
       with CC select
              C <= '0' when         "0000" |
                                    "0001" |
                                    "0011" |
                                    "0100" |
                                    "0101" |
                                    "0110" |
                                    "0111"|
                                    "1000" |
                                    "1001",
                           '1' when      others;
       with CD select
              D      <= '0' when           "0000" |
                                    "0010" |
                                    "0011" |
                                    "0101" |
                                    "0110" |
                                    "1000" |
                                    "1001",
                           '1' when              others;
       with CE select
              E <= '0' when         "0000" |
                                    "0010" |
```

```
                                    "0110"|
                                    "1000",
                              '1' when              others;
       with CF select
              F <= '0' when         "0000" |
                                    "0100" |
                                    "0101" |
                                    "0110"|
                                    "1000" |
                                    "1001",
                              '1' when              others;
       with CG select
              G <= '0' when         "0010" |
                                    "0011" |
                                    "0100" |
                                    "0101" |
                                    "0110"|
                                    "1000" |
                                    "1001",
                              '1' when              others;

end Truth_table;
```

Código .ucf para asignar los pines de la placa *Spartan 3E-500*

```
NET "CD" LOC = R17;
NET "CE" LOC = N17;
NET "H" LOC = J14;
NET "CLK" LOC = "B8";
NET "A" LOC = L18;
NET "B" LOC = F18;
NET "C" LOC = D17;
NET "D" LOC = D16;
NET "E" LOC = G14;
NET "F" LOC = J17;
NET "G" LOC = H14;
```

Anexo 2 Códigos. UART

Top_UART

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY UART_module IS
       PORT (
              OSC_CLK_in            :       IN  STD_LOGIC;
              CD                            :       IN  STD_LOGIC;
              Enable_TX             :       IN  STD_LOGIC;
              TX_IN                 :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);

              ST                            :       IN      STD_LOGIC;
              Serial_out            :       OUT  STD_LOGIC;
              ET                            :       OUT     STD_LOGIC;

              Enable_RX             :       IN  STD_LOGIC;
          Serial_in          :       IN  STD_LOGIC;
              RX_OUT                :       OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
              Error                 :       OUT STD_LOGIC;
              New_Data              :       OUT STD_LOGIC;
              CLK_1Hz_SQUARED       :       OUT STD_LOGIC -- The 1 Hz LED to show that
the system works
       );
END UART_module;


ARCHITECTURE structural OF UART_module IS


-- The Four components

COMPONENT BAUD_RATE_GENERATOR IS
       PORT(
              CD,CE                 :       IN  std_logic;
              OSC_CLK_in            :       IN  std_logic;
              OSC_CLK_out           :       OUT std_logic;
              RX_CLK                :       OUT std_logic;
              TX_CLK                :       OUT std_logic;
              CLK_100Hz             :       OUT std_logic;
              CLK_1Hz_SQUARED       :       OUT std_logic
       );
END COMPONENT;

COMPONENT deboucing_filter IS
       Port (
              CLK     : IN    STD_LOGIC;
              PB_L    : IN    STD_LOGIC;
              CD      : IN STD_LOGIC;
              QA              : OUT   STD_LOGIC;
              QB              : OUT   STD_LOGIC
       );
END COMPONENT;

COMPONENT Transmitter_unit IS
       PORT (
              CLK             :       IN      STD_LOGIC;
              CD                    :       IN      STD_LOGIC;
              TX_IN           :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
              ST                    :       IN      STD_LOGIC;
              Enable_TX       :       IN      STD_LOGIC;
              ET                    :       OUT     STD_LOGIC;
              Serial_out      :       OUT STD_LOGIC
       );
END COMPONENT;

COMPONENT Receiver_unit IS
       PORT (
              CLK                   :       IN  STD_LOGIC;
              CD                    :       IN  STD_LOGIC;
              Enable_RX       :       IN  STD_LOGIC;
```

```
          Serial_in   :      IN  STD_LOGIC;
               RX_OUT         :      OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
               Error          :      OUT STD_LOGIC;
               New_Data       :      OUT STD_LOGIC
       );
END COMPONENT;


-- The signals to connect the modules:
       SIGNAL TX_CLK          :      STD_LOGIC;
       SIGNAL RX_CLK          :      STD_LOGIC;
       SIGNAL CLK_100Hz       :      STD_LOGIC;
       SIGNAL OSC_CLK_out     :      STD_LOGIC;
       SIGNAL ST_Pulse                :      STD_LOGIC;    -- Only one clock pulse (10 ms
- 100 Hz )
       SIGNAL ST_Level                :      STD_LOGIC;       -- At '1' as long as the
user is clikling
       SIGNAL ST_L                    :      STD_LOGIC;     -- The debouncing filter was
designed with an active low input
       --SIGNAL Serial_in             :      STD_LOGIC;
       --SIGNAL Serial_out            :      STD_LOGIC;

BEGIN

-- Instantiation of the components:
Chip1 : BAUD_RATE_GENERATOR
       PORT MAP (
-- from component name        => to signal or port name
               CD                              =>      CD,
               CE                              =>      '1',
               OSC_CLK_in              =>      OSC_CLK_in,
               OSC_CLK_out             =>      OSC_CLK_out,
               RX_CLK                  =>      RX_CLK,
               TX_CLK                  =>      TX_CLK,
               CLK_100Hz               =>      CLK_100Hz,
               CLK_1Hz_SQUARED         =>      CLK_1Hz_SQUARED
       );

Chip2 : deboucing_filter
       PORT MAP (
-- from component name        => to signal or port name
               CLK                             =>      CLK_100Hz,
               PB_L                    =>      ST_L,
               CD                      =>      CD,
               QA                              =>      ST_Pulse,
               QB                              =>      ST_Level
       );

Chip3 : Transmitter_unit
-- from component name        => to signal or port name
       PORT MAP(
               CLK                     =>              TX_CLK,
               CD                              =>              CD,
               TX_IN                   =>              TX_IN,
               ST                              =>              ST_Pulse,

               Enable_TX               =>              Enable_TX,
               ET                              =>              ET,
               Serial_out              =>              Serial_out
       );

Chip4 : Receiver_unit
-- from component name        => to signal or port name
       PORT MAP(
               CLK                     =>              RX_CLK,
               CD                              =>              CD,
               Enable_RX               =>              Enable_RX,
          Serial_in            =>              Serial_in,
               RX_OUT                  =>              RX_OUT,
               Error                   =>              Error,
               New_Data                =>              New_Data
       );


-- Extra logic
```

```
ST_L <= ST;


END structural ;
```

Baud_Rate_Generator.vhd

```
LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY BAUD_RATE_GENERATOR IS
      PORT(    CD,CE                          :       IN  std_logic;
                       OSC_CLK_in                     :       IN  std_logic;
                       OSC_CLK_out                    :       OUT std_logic;
                       RX_CLK                         :       OUT std_logic;
                       TX_CLK                         :       OUT std_logic;
                       CLK_100Hz                      :       OUT std_logic;
                       CLK_1Hz_SQUARED                :       OUT std_logic
          );
END BAUD_RATE_GENERATOR ;

ARCHITECTURE schematic OF BAUD_RATE_GENERATOR  IS
-- Components
       COMPONENT Freq_Div_326 IS
              PORT(   CD,CLK,CE      :       IN  std_logic;
                              TC326          :       OUT std_logic
              );
       END COMPONENT;

       COMPONENT Freq_Div_8 IS
              PORT(   CD,CLK,CE      :       IN  std_logic;
                              TC8                    :       OUT std_logic
              );
       END COMPONENT;

       COMPONENT Freq_Div_100 IS
              PORT(   CD,CLK,CE      :       IN  std_logic;
                              TC100          :       OUT std_logic
              );
       END COMPONENT;

       COMPONENT Freq_Div_96 IS
              PORT(   CD,CLK,CE      :       IN  std_logic;
                              TC96           :       OUT std_logic
              );
       END COMPONENT;

       COMPONENT T_Flip_Flop IS
      Port (
                       CLK     : IN    STD_LOGIC;
                       CD              : IN    STD_LOGIC;
                       T               : IN    STD_LOGIC;
                       Q               : OUT   STD_LOGIC
              );
       END COMPONENT;

-- Signals for connecting components together (just the internal wires)
       SIGNAL OSC_CLK                                 :       std_logic;
       SIGNAL RX_CLK_x2, TX_CLK_x2    :       std_logic;
       SIGNAL CLK_100Hz_x2                    :       std_logic;
       SIGNAL CLK_1Hz_x2                              :       std_logic;

       BEGIN
-- Instantiation of components

       CHIP1   :       Freq_Div_326
       PORT MAP (
-- from component name        => to signal or port name
                       CLK             => OSC_CLK,
                       CD              => CD,
                       CE              => CE,
```

```
                        TC326   => RX_CLK_x2
                );

        CHIP2   :  Freq_Div_8
                PORT MAP (
-- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        CE              => RX_CLK_x2,
                        TC8     => TX_CLK_x2
                );

        CHIP3   :       Freq_Div_100
                PORT MAP (
-- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        CE              => TX_CLK_x2,
                        TC100   => CLK_100Hz_x2
                );

        CHIP4   :       Freq_Div_96
                PORT MAP (
-- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        CE              => CLK_100Hz_x2,
                        TC96    => CLK_1Hz_x2
                );

        Chip5   : T_Flip_Flop
                PORT MAP (
        -- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        T               => RX_CLK_x2,
                        Q               => RX_CLK
                );

        Chip6   : T_Flip_Flop
                PORT MAP (
        -- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        T               => TX_CLK_x2,
                        Q               => TX_CLK
                );

        Chip7   : T_Flip_Flop
                PORT MAP (
        -- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        T               => CLK_100Hz_x2,
                        Q               => CLK_100Hz
                );

        Chip8   : T_Flip_Flop
                PORT MAP (
        -- from component name          => to signal or port name
                        CLK             => OSC_CLK,
                        CD              => CD,
                        T               => CLK_1Hz_x2,
                        Q               => CLK_1Hz_SQUARED
                );

-- connections and logic between components

        -- The circuit's signals that have to be connected to input ports
        OSC_CLK <= OSC_CLK_in;

        -- The output ports that have to be connected to internal signals
        OSC_CLK_out     <= OSC_CLK;

END schematic ;
```

Freq_div_326.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Freq_Div_326 IS
              PORT(   CD,CLK,CE      :       IN  std_logic;
                              TC326          :       OUT std_logic
              );

END Freq_Div_326;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF Freq_Div_326 IS

CONSTANT Max_Count    :  STD_LOGIC_VECTOR(8  DOWNTO  0)  :=  "101000101";  --  325  -->
terminal_count after 326 states
CONSTANT Reset        : STD_LOGIC_VECTOR(8 DOWNTO 0) := "000000000";

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(8 DOWNTO 0);

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                      -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- ESS state_registercombinational system for calculating next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
              IF(present_state < Max_Count ) THEN
                     future_state <= present_state + 1 ;
              ELSE
                     future_state <= Reset;
              END IF;
       ELSE
              future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

------------------------- CS_2: combinational system for calculating extra outputs
-------------------------- and outputing the present state (the actual count)

TC326 <= '1' WHEN (present_state = Max_count AND CE = '1') ELSE '0';  --terminal count



END FSM_like;
```

Freq_div_8.vhd

```
LIBRARY ieee;
```

```
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Freq_Div_8 IS
               PORT(  CD,CLK,CE      :       IN  std_logic;
                              TC8                    :       OUT std_logic
               );

END Freq_Div_8;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF Freq_Div_8 IS

CONSTANT Max_Count    : STD_LOGIC_VECTOR(2 DOWNTO 0) := "111"; -- terminal_count after 8
states
CONSTANT Reset        : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                       -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- ESS state_registercombinational system for calculating next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
              IF(present_state < Max_Count ) THEN
                     future_state <= present_state + 1 ;
              ELSE
                     future_state <= Reset;
              END IF;
       ELSE
              future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

------------------------- CS_2: combinational system for calculating extra outputs
------------------------- and outputing the present state (the actual count)

TC8 <= '1' WHEN (present_state = Max_count AND CE = '1') ELSE '0';  --terminal count



END FSM_like;
```

Freq_div_100.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Freq_Div_100 IS
               PORT(  CD,CLK,CE      :       IN  std_logic;
                              TC100          :       OUT std_logic
               );

END Freq_Div_100;
```

```
                    -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF Freq_Div_100 IS

CONSTANT Max_Count    :  STD_LOGIC_VECTOR(6  DOWNTO  0)  :=  "1100011";  --  terminal_count
after 99 states
CONSTANT Reset        : STD_LOGIC_VECTOR(6 DOWNTO 0) := "0000000";

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(6 DOWNTO 0);

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                     -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------  ESS  state_registercombinational  system  for  calculating  next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
              IF(present_state < Max_Count ) THEN
                     future_state <= present_state + 1 ;
              ELSE
                     future_state <= Reset;
              END IF;
       ELSE
              future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

------------------------- CS_2: combinational system for calculating extra outputs
--------------------------- and outputing the present state (the actual count)

TC100 <= '1' WHEN (present_state = Max_count AND CE = '1') ELSE '0';  --terminal count



END FSM_like;
```

Freq_div_96.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Freq_Div_96 IS
              PORT(  CD,CLK,CE      :       IN  std_logic;
                            TC96            :       OUT std_logic
              );

END Freq_Div_96;

                    -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF Freq_Div_96 IS

CONSTANT Max_Count    :  STD_LOGIC_VECTOR(7  DOWNTO  0)  :=  "10111111";  --  terminal_count
after 96 states
CONSTANT Reset        : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000";
```

```
-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
       BEGIN

       IF CD = '1' THEN                      -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- ESS state_registercombinational system for calculating next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
              IF(present_state < Max_Count ) THEN
                     future_state <= present_state + 1 ;
              ELSE
                     future_state <= Reset;
              END IF;
       ELSE
              future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

-------------------------- CS_2: combinational system for calculating extra outputs
-------------------------- and outputing the present state (the actual count)

TC96 <= '1' WHEN (present_state = Max_count AND CE = '1') ELSE '0';  --terminal count



END FSM_like;
```

T_flip_flop.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY T_Flip_Flop IS
      Port (   CLK     : IN    STD_LOGIC;
               CD      : IN    STD_LOGIC;
               T       : IN    STD_LOGIC;
               Q       : OUT   STD_LOGIC
       );

END T_Flip_Flop;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF T_Flip_Flop IS


CONSTANT Reset : STD_LOGIC := '0';

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC := Reset;
  -- This thing of initialising these signals to the "Reset" state,
  -- is only an issue for the functional simulator. Once the circuit
  -- is synthesised, this thing is completely irrelevant.

BEGIN
------------------------------ the only clocked block : the state register
```

```
state_register: PROCESS (CD, CLK)
        BEGIN

        IF CD = '1' THEN                        -- reset counter
                present_state <= Reset;
        ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
                present_state <= future_state;
        END IF;

END PROCESS state_register;

------------------------- next state logic
-- A T flip flop invert the output when T = 1, and do nothing when T = 0

CS_1: PROCESS (present_state, T)
        BEGIN
        IF T = '1' THEN
                        future_state <= NOT (present_state);
        ELSE
                future_state <= present_state;
        END IF;

END PROCESS CS_1;

------------------------- CS_2: combinational system for calculating extra outputs
-- Very simple in this case, a buffer.
Q <= present_state;


END FSM_like;
```

Debouncing_filter.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY deboucing_filter IS
      Port (   CLK     : IN    STD_LOGIC;
                               PB_L            : IN    STD_LOGIC;
                               CD      : IN STD_LOGIC;
                               QA              : OUT   STD_LOGIC;
                               QB              : OUT   STD_LOGIC
       );

END deboucing_filter;


ARCHITECTURE FSM_like OF deboucing_filter IS


       TYPE    State_type    IS    (Idle,    One_zero_detected,    Two_zeroes_detected,
Three_zeroes_detected, Key_presed, One_high);

       ATTRIBUTE syn_encoding : STRING;
       ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";

       SIGNAL present_state, future_state   : State_type ;

BEGIN


state_register: PROCESS (CD, CLK)
        BEGIN

        IF CD = '1' THEN                        -- reset counter
                present_state <= idle;
        ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
                present_state <= future_state;
        END IF;

END PROCESS state_register;
```

```
------------------------- CC1: Combinational system for calculating next state

CC_1: PROCESS (present_state, PB_L)
        BEGIN

                CASE present_state IS

                        WHEN Idle =>
                           IF (PB_L = '1') THEN
                                future_state <= Idle;
                                ELSE
                                        future_state <= One_zero_detected;
                                END IF;

                        WHEN One_zero_detected =>
                                IF (PB_L = '1') THEN
                                        future_state <= Idle;
                                ELSE
                                        future_state <= Two_zeroes_detected ;
                                END IF;

                        WHEN Two_zeroes_detected =>
                           IF (PB_L = '1') THEN
                              future_state <= idle;
                                ELSE
                                        future_state <= Three_zeroes_detected ;
                                END IF;

                        WHEN Three_zeroes_detected =>
                                        future_state <= key_presed ;

                        WHEN Key_presed =>
                           IF (PB_L = '0') THEN
                              future_state <= Key_presed;
                                ELSE
                                        future_state <= One_high ;
                                END IF;

                        WHEN One_high =>
                           IF (PB_L = '1') THEN
                              future_state <= Idle;
                                ELSE
                                        future_state <= Key_presed ;
                                END IF;

                END CASE ;

END PROCESS CC_1;

------------------------- CS_2: combinational system for calculating extra outputs
------------------------- and outputing the present state (the actual count)

CC_2: PROCESS (present_state)
        BEGIN

                CASE present_state IS
                                WHEN Idle =>
                                        QA <= '0';
                                        QB <= '0';
                                WHEN One_zero_detected =>
                                        QA <= '0';
                                        QB <= '0';
                                WHEN Two_zeroes_detected =>
                                        QA <= '0';
                                        QB <= '0';
                                WHEN Three_zeroes_detected =>
                                        QA <= '1';
                                        QB <= '1';
                                WHEN Key_presed =>
                                        QA <= '0';
                                        QB <= '1';
                                WHEN One_high =>
                                        QA <= '0';
                                        QB <= '1';
                END CASE ;
```

```
END PROCESS CC_2;

-- Place other logic if necessary

END FSM_like;
```

Transmitter_unit.vhd

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY Transmitter_unit IS

PORT (
               CLK            :       IN      STD_LOGIC;
               CD                     :       IN      STD_LOGIC;
               TX_IN          :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
               ST                     :       IN      STD_LOGIC;
               Enable_TX      :       IN      STD_LOGIC;
               ET                     :       OUT     STD_LOGIC;
               Serial_out     :       OUT STD_LOGIC
               );
END Transmitter_unit;


ARCHITECTURE structural OF Transmitter_unit IS


COMPONENT  transmitter_datapath IS
       PORT (
               CLK                    :       IN STD_LOGIC;
               LDD                    :       IN STD_LOGIC;
               CE                     :       IN STD_LOGIC;
               CD                     :       IN STD_LOGIC;
               TX_IN          :       IN STD_LOGIC_VECTOR (7 DOWNTO 0);
               S                      :       IN STD_LOGIC_VECTOR (1 DOWNTO 0);
               F                      :       IN STD_LOGIC_VECTOR (1 DOWNTO 0);
               TC8                    :       OUT STD_LOGIC;
               Serial_out     :       OUT STD_LOGIC
               );
       END COMPONENT;

COMPONENT transmitter_control_unit IS
       PORT(
                       CLK, CD, Enable_TX, ST:       IN  STD_LOGIC;
                       TC8                                             :       IN  STD_LOGIC;
                   F                           :   OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
                       S                               :     OUT  STD_LOGIC_VECTOR(1  DOWNTO
0);
                       LDD                                             :       OUT std_logic;
                       ET                                              :       OUT std_logic;
                       CE                                      :       OUT std_logic

        );
       END COMPONENT;

   SIGNAL  F,S                : STD_LOGIC_VECTOR (1 DOWNTO 0);
   SIGNAL  TC8, LDD, CE       : STD_LOGIC;

BEGIN

Chip1  :       transmitter_control_unit
           PORT MAP (
                       CLK        => CLK,
                       CD         => CD,
                       Enable_TX => Enable_TX,
                       ST         => ST,
            TC8        => TC8,
            F          => F,
            S          => S,
            CE         => CE,
            LDD        => LDD,
                       ET         => ET
```

```
                          );

Chip2   :       transmitter_datapath
           PORT MAP (
                     CLK             => CLK,
                     LDD          => LDD,
                     CE           => CE,
                     CD           => CD,
                     TX_IN        => TX_IN,
                     S            => S,
                     F            => F,
                     TC8          => TC8,
                     Serial_out      => Serial_out
                );


END structural ;
```

Transmitter_control_unit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY transmitter_control_unit IS
      Port (
           CLK               :       IN      STD_LOGIC;
              CD             :       IN      STD_LOGIC;
              Enable_TX      :       IN      STD_LOGIC;
              ST                     :       IN      STD_LOGIC;
              TC8            :       IN      STD_LOGIC;
              F              :       OUT     STD_LOGIC_VECTOR(1 DOWNTO 0);
              S                      :       OUT     STD_LOGIC_VECTOR(1 DOWNTO 0);
              LDD            :       OUT     STD_LOGIC;
              CE             :       OUT     STD_LOGIC;
              ET             :       OUT     STD_LOGIC
       );

END transmitter_control_unit;


ARCHITECTURE FSM_like OF transmitter_control_unit IS


       TYPE State_type IS (Idle, Load_Data_Reg, Load_Shift_Reg, Send_Start_bit, Send_D0,
Send_D1_to_D7, Send_EPB, Send_Stop_bit, Wait_not_ready) ;

       ATTRIBUTE syn_encoding : STRING;
       ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";

       SIGNAL present_state, future_state   : State_type ;


CONSTANT Reset        : State_type := Idle; -- The fisrt state.

BEGIN


state_register: PROCESS (CD,CLK)
       BEGIN

       IF CD = '1' THEN                     -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------ CC1: Combinational system for calculating next state

CC_1: PROCESS (present_state, TC8, Enable_TX, ST)
```

```
        BEGIN
                CASE present_state IS
                        WHEN Idle =>
                            if (ST = '1' and Enable_TX = '1') THEN
                                      future_state <= Load_Data_Reg;
                            else
                               future_state <= Idle ;
                                end if;

                        WHEN Load_Data_Reg =>
                               future_state <= Load_Shift_Reg;

                        WHEN Load_Shift_Reg =>
                               future_state <= Send_Start_bit ;

                        WHEN Send_Start_bit =>
                               future_state <= Send_D0 ;

                        WHEN Send_D0 =>
                               future_state <= Send_D1_to_D7 ;

                        WHEN Send_D1_to_D7 =>
                            if (TC8 = '0') THEN
                                future_state <= Send_D1_to_D7;
                            else
                                    future_state <= Send_EPB ;
                                end if;

                        WHEN Send_EPB =>
                               future_state <= Send_Stop_bit ;

                        WHEN Send_Stop_bit =>
                               future_state <= Wait_not_ready ;

                        WHEN Wait_not_ready =>
                            if (ST = '1') THEN
                                future_state <= Wait_not_ready;
                            else
                                future_state <= Idle ;
             end if;

                        END CASE ;

END PROCESS CC_1;

------------- CS_2: combinational system for calculating outputs at every state (Moore
machine)
CC_2: PROCESS (present_state)
        BEGIN

                CASE present_state IS
                                WHEN Idle =>
                                       LDD <= '0';
                                       F       <= "00";
                                       S       <= "00";
                                       CE      <= '0';
                                       ET      <= '0';

                                WHEN Load_Data_Reg =>
                                       LDD <= '1';
                                       F       <= "00";       -- Channe1 Ch0 --> Marking at
level '1'
                                       S       <= "00";
                                       CE      <= '0';
                                       ET      <= '0';

                                WHEN Load_Shift_Reg =>
                                       LDD <= '0';
                                       F       <= "00";
                                       S       <= "11";       -- Load the data register
                                       CE      <= '0';
                                       ET      <= '0';

                                WHEN Send_Start_bit =>
                                       LDD <= '0';
                                       F       <= "01";     -- Channe1 Ch1
```

```
                                    S       <= "00";
                                    CE      <= '0';
                                    ET      <= '0';

                                WHEN Send_D0 =>
                                    LDD <= '0';
                                    F       <= "10";      -- Channe1 Ch2 is the RSO
                                    S       <= "01";      -- and prepare to shift data on
the next rising edge
                                    CE      <= '1';        --  allow  the  counter  start
counting
                                    ET      <= '0';

                                WHEN Send_D1_to_D7 =>
                                    LDD <= '0';
                                    F       <= "10";
                                    S       <= "01";      -- shift data while the counter
is not full
                                    CE      <= '1';
                                    ET      <= '0';

                                WHEN Send_EPB =>
                                    LDD <= '0';
                                    F       <= "11";       --   Ch3   to   send   the   Even
parity Bit
                                    S       <= "00";
                                    CE      <= '0';
                                    ET      <= '0';

                                WHEN Send_Stop_bit =>
                                    LDD <= '0';
                                    F       <= "00";    -- Ch0 to send the stop bit and
marking at '1'
                                    S       <= "00";
                                    CE      <= '0';
                                    ET      <= '1';    -- at the same state generate a
pulse of "end of transmission"

                                WHEN Wait_not_ready =>
                                    LDD <= '0';
                                    F       <= "00";
                                    S       <= "00";
                                    CE      <= '0';
                                    ET      <= '0';
            END CASE ;


END PROCESS CC_2;

-- Place other logic if necessary

END FSM_like;
```

Transmitter_datapath.vhd

```
ENTITY Transmitter_datapath IS

PORT (
            CLK             :       IN STD_LOGIC;
            CD                      :       IN STD_LOGIC;
            CE                      :       IN STD_LOGIC;
            LDD                     :       IN STD_LOGIC;
            TX_IN           :       IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            S                       :       IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            F                       :       IN STD_LOGIC_VECTOR (1 DOWNTO 0);
            TC8             :       OUT STD_LOGIC;
            Serial_out      :       OUT STD_LOGIC
            );

END Transmitter_datapath ;


ARCHITECTURE schematic OF Transmitter_datapath IS
```

```
COMPONENT  MUX4 IS
       PORT (
              Ch3, Ch2, Ch1, Ch0     :       IN STD_LOGIC;
     E                  :    IN STD_LOGIC;
              S                                     :       IN STD_LOGIC_VECTOR (1 DOWNTO
0);
              Y                                     :       OUT STD_LOGIC
       );
       END COMPONENT;

COMPONENT parity_generator_8bit IS
       PORT (
              X_in    :       IN STD_LOGIC_VECTOR(7 downto 0);
              EPB     :       OUT STD_LOGIC
       );
       END COMPONENT;

COMPONENT shift_register_10bit IS
      Port (
              CLK            :       IN      STD_LOGIC;
              CD             :       IN      STD_LOGIC;
              S              :       IN      STD_LOGIC_VECTOR(1 DOWNTO 0);
              RSI     :      IN      STD_LOGIC;
              LSI            :       IN      STD_LOGIC;
              Q              :       OUT     STD_LOGIC_VECTOR(9 DOWNTO 0);
              D_IN    :      IN STD_LOGIC_VECTOR(9 DOWNTO 0)
       );
       END COMPONENT ;


COMPONENT data_register_8bits IS
     PORT (
       CLK            :       IN      STD_LOGIC;
              CD             :       IN      STD_LOGIC;
           LD         :       IN      STD_LOGIC;
              Q              :       OUT     STD_LOGIC_VECTOR(7 DOWNTO 0);
              D_in    :      IN STD_LOGIC_VECTOR(7 DOWNTO 0)
       );
       END COMPONENT ;

COMPONENT counter_mod8 IS
       PORT(
       CD,CLK,CE      :   IN  std_logic;
         TC8          :   OUT std_logic
          );
       END COMPONENT ;


-- Signals
   SIGNAL      B                     : STD_LOGIC_VECTOR (7 DOWNTO 0);
   SIGNAL      Q                     : STD_LOGIC_VECTOR (9 DOWNTO 0);
   SIGNAL      EPB, RSO       : STD_LOGIC;

BEGIN

Chip1  :      data_register_8bits
           PORT MAP (
                     LD             => LDD,
                     D_in    => TX_IN,
            CD      => CD,
            CLK     => CLK,
            Q       => B
                  );

Chip2  :      parity_generator_8bit
           PORT MAP (
              -- from component name        => to signal or port name
                     X_in    => B,
                     EPB            => EPB
                  );

Chip3  :  counter_mod8
           PORT MAP (
              -- from component name        => to signal or port name
                     CD             => CD,
                     CLK            => CLK,
```

```
                    CE              => CE,
                    TC8     => TC8
                );

Chip4   :       shift_register_10bit
            PORT MAP (
                -- from component name        => to signal or port name
                    CLK  => CLK,
                    CD       => CD,
                    S        => S,
                    RSI  => '0',
                    LSI      => '0',
                    Q(9 downto 3) => Q(9 downto 3),
                    Q(2) => RSO,
                    Q(1 downto 0) => Q(1 downto 0),
                    D_IN (9 downto 2)  => B,
                    D_IN (1 downto 0) => "00"
                );

Chip5   :       MUX4
            PORT MAP (
                -- from component name        => to signal or port name
                    Ch0             => '1',
                    Ch1             => '0',
                    Ch2             => RSO,
                    ch3     => EPB,
                    E       => '1',
                    S        => F,
          Y       => Serial_out
                    );

END schematic ;
```

Receiver_unit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY Receiver_unit IS
       PORT (
              CLK                   :       IN  STD_LOGIC;
              CD                    :       IN  STD_LOGIC;
              Enable_RX      :       IN  STD_LOGIC;
              Serial_in    :  IN  STD_LOGIC;
              RX_OUT         :       OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
              Error          :       OUT STD_LOGIC;
              New_Data       :       OUT STD_LOGIC
       );
END Receiver_unit;


ARCHITECTURE structural OF Receiver_unit IS

COMPONENT receiver_control_unit IS
       PORT (
                    CLK                     : IN    STD_LOGIC;
                    CD                              : IN    STD_LOGIC;
                    Start_Bit               : IN    STD_LOGIC;
                    Enable_RX               : IN    STD_LOGIC;
                    TC4                             : IN    STD_LOGIC;
                    TC8                             : IN    STD_LOGIC;
                    TC10                    : IN    STD_LOGIC;
                    EP_Check                : IN    STD_LOGIC;
                    Stop_Bit                : IN    STD_LOGIC;
                    S                               : OUT     STD_LOGIC_VECTOR (1 DOWNTO
0);
                    LDD                             : OUT   STD_LOGIC;
                    CE_Mod4                 : OUT   STD_LOGIC;
                    CE_Mod8                 : OUT   STD_LOGIC;
                    Clear_Counters : OUT   STD_LOGIC;
                    Error                   : OUT   STD_LOGIC;
                    New_Data                : OUT   STD_LOGIC
```

```
                );
END COMPONENT;

COMPONENT Receiver_datapath IS
        PORT (
                Serial_in               :       IN STD_LOGIC;
                S                               :       IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                LDD                             :       IN STD_LOGIC;
                CE_Mod4                 :       IN STD_LOGIC;
                Clear_Counters :        IN STD_LOGIC;
                CE_Mod8                 :       IN STD_LOGIC;
                CD                              :       IN STD_LOGIC;
                CLK                             :       IN STD_LOGIC;
                RX_OUT                  :       OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
                EP_Check                :       OUT STD_LOGIC;
                Stop_Bit                :       OUT STD_LOGIC;
                TC4                             :       OUT STD_LOGIC;
                TC8                             :       OUT STD_LOGIC;
                TC10                    :       OUT STD_LOGIC
                );
END COMPONENT;

        SIGNAL Start_Bit                : STD_LOGIC;
        SIGNAL Stop_Bit                         : STD_LOGIC;
        SIGNAL EP_Check                 : STD_LOGIC;
        SIGNAL TC8                              : STD_LOGIC;
        SIGNAL TC4                              : STD_LOGIC;
        SIGNAL TC10                     : STD_LOGIC;
        SIGNAL LDD                              : STD_LOGIC;
        SIGNAL S                                : STD_LOGIC_VECTOR (1 DOWNTO 0);
        SIGNAL CE_Mod4                  : STD_LOGIC;
        SIGNAL CE_Mod8                  : STD_LOGIC;
        SIGNAL Clear_Counters : STD_LOGIC;

BEGIN

Chip1 : receiver_control_unit
        PORT MAP (
                Enable_RX               => Enable_RX,
                Start_Bit               => Start_Bit,
                Stop_Bit                => Stop_Bit,
                EP_Check                => EP_Check,
                TC4                     => TC4,
                TC8                     => TC8,
                TC10                    => TC10,
                CD                              => CD,
                CLK                     => CLK,
                LDD                     => LDD,
                S                               => S,
                CE_Mod4                 => CE_Mod4,
                CE_Mod8                 => CE_Mod8,
                Clear_Counters          => Clear_Counters,
                Error                   => Error,
                New_Data                => New_Data
        );

Chip2 : Receiver_datapath
        PORT MAP (
                Serial_in               => Serial_in,
                LDD                             => LDD,
                S                               => S,
                CE_Mod4                 => CE_Mod4,
                CE_Mod8                 => CE_Mod8,
                Clear_Counters          => Clear_Counters,
                CD                              => CD,
                CLK                     => CLK,
                RX_OUT                  => RX_OUT,
                Stop_Bit                => Stop_Bit,
                EP_Check                => EP_Check,
                TC4                     => TC4,
                TC8                     => TC8,
                TC10                    => TC10
        );

-- Extra logic
Start_Bit <= Serial_in;
```

```
END structural ;
```

Receiver_control_unit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;


ENTITY receiver_control_unit IS
      Port (
               Start_Bit      : IN    STD_LOGIC;
               CLK                    : IN    STD_LOGIC;
               CD                      : IN   STD_LOGIC;
               Enable_RX          : IN       STD_LOGIC;
               EP_check           : IN       STD_LOGIC;
               Stop_Bit               : IN    STD_LOGIC;
               TC4,TC8,TC10   : IN    STD_LOGIC;
               S                      : OUT     STD_LOGIC_VECTOR (1 DOWNTO 0);
               LDD             : OUT  STD_LOGIC;
               CE_MOD4            : OUT      STD_LOGIC;
               CE_MOD8        : OUT   STD_LOGIC;
               Clear_Counters : OUT   STD_LOGIC;
               Error                  : OUT   STD_LOGIC;
               New_Data               : OUT   STD_LOGIC
       );

END receiver_control_unit;

                     -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF receiver_control_unit IS

-- Internal wires
                             -- State signals declaration
                             -- A BCD counter will consist of 10 diferent states
       TYPE   State_type   IS   (Idle,   wait_4cycles,   wait_8cycles,   sample_serial_in,
last_sample, test_parity,
                           test_stop_bit, error_signal, load_data_reg, data_ready) ;
--------> This is important: specifying to the synthesiser the code for the states
       ATTRIBUTE syn_encoding : STRING;
       ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";
              -- (It may be:"sequential" (binary); "gray"; "one-hot", etc.
       SIGNAL present_state, future_state   : State_type ;

-- Constants for special states (the first and the last state)
CONSTANT Reset        : State_type := Idle; -- The fisrt state. This is another name for
the state Idle
                                                                                --    so
that the state registes doesn't have to be modified

BEGIN
---------------------- State Register: the only clocked block.
---------------------- The "memory" of the system (future events will depend on past
events
state_register: PROCESS (CD,CLK, future_state)
       BEGIN

       IF CD = '1' THEN                     -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- CC1: Combinational system for calculating next state

CC_1: PROCESS (present_state, Stop_bit, EP_check, TC10, TC8, TC4,Enable_RX,Start_bit)
       BEGIN
                     CASE present_state IS
                            WHEN Idle =>
                                   IF (Start_bit = '0' and Enable_RX = '1') THEN
```

```
                                    future_state <= wait_4cycles;
                                ELSE
                                    future_state <= Idle  ;
                                END IF;

                            WHEN wait_4cycles =>
                                IF (TC4 = '1' and Start_bit = '0') THEN
                                   future_state <= wait_8cycles;
                                ELSIF (TC4 = '1' and Start_bit = '1')THEN
                                   future_state <= Idle;
                                ELSE
                                      future_state <= wait_4cycles;
                                   END IF;

                            WHEN wait_8cycles =>
                                IF (TC10 = '1') THEN
                                   future_state <= last_sample;
                                ELSIF (TC8 = '1') THEN
                        future_state <= sample_serial_in;
                                ELSE
                                   future_state <= wait_8cycles;
                                   END IF;

                            WHEN sample_serial_in =>
                                IF (TC10 = '1') THEN
                                          future_state <= test_parity;
                                   ELSE
                                          future_state <= wait_8cycles;
                                   END IF;

                            WHEN last_sample =>
                                          future_state <= test_parity;

                            WHEN test_parity =>
                                IF (EP_check = '1') THEN
                                  future_state <= error_signal;
                                ELSE
                                  future_state <= test_stop_bit;
                                   END IF;

                            WHEN error_signal =>
                                  future_state <= Idle;

                            WHEN test_stop_bit =>
                                IF (Stop_Bit = '0') THEN
                                      future_state <= error_signal;
                                   ELSE
                                      future_state <= load_data_reg;
                                   END IF;

                            WHEN load_data_reg =>
                                   future_state <= data_ready ;

                            WHEN data_ready =>
                                   future_state <= Idle ;

                        END CASE ;
END PROCESS CC_1;

-------------------------- CS_2: combinational system for calculating extra outputs
-------------------------- and outputing the present state (the actual count)

CC_2: PROCESS (present_state)
       BEGIN
                CASE present_state IS
                            WHEN Idle =>
                                   S                    <= "00";
                                   LDD             <= '0';
                                   CE_MOD4            <= '0';
                                   CE_MOD8         <= '0';
                                   Clear_Counters <= '0';
                                   Error                   <= '0';
                                   New_Data                <= '0';

                            WHEN wait_4cycles =>
                                S               <= "00";
```

```
            LDD             <= '0';
            CE_MOD4             <= '1';
            CE_MOD8         <= '0';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN wait_8cycles =>
            S               <= "00";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '1';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN sample_serial_in =>
            S               <= "01";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '1';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN last_sample =>
            S             <= "01";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8          <= '0';
            Clear_Counters<= '0';
            Error                       <= '0';
            New_Data                    <= '0';

        WHEN test_parity =>
            S               <= "00";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '0';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN error_signal =>
            S               <= "00";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '0';
            Clear_Counters <= '1';
            Error                   <= '1';
            New_Data                <= '0';

        WHEN test_stop_bit =>
            S               <= "00";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '0';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN load_data_reg =>
            S                    <= "00";
            LDD             <= '1';
            CE_MOD4             <= '0';
            CE_MOD8         <= '0';
            Clear_Counters <= '0';
            Error                   <= '0';
            New_Data                <= '0';

        WHEN data_ready =>
            S                   <= "00";
            LDD             <= '0';
            CE_MOD4             <= '0';
            CE_MOD8         <= '0';
            Clear_Counters <= '1';
```

```
                                    Error                <= '0';
                                    New_Data             <= '1';

                        END CASE ;

END PROCESS CC_2;

-- Place other logic if necessary

END FSM_like;
```

Receiver_datapath.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;

ENTITY Receiver_datapath IS
       PORT(
              Serial_in,CD,CLK      :       IN     std_logic;
              LDD                                  :       IN     std_logic;

              CE_Mod4,CE_Mod8               :       IN     std_logic;
              Clear_Counters        :       IN     std_logic;
              s                                    :       IN     std_logic_vector(1
downto 0);
              RX_out                        :       OUT std_logic_vector(7 downto 0);
              EP_Check,Stop_bit     :       OUT std_logic;

              TC4,TC8,TC10          :       OUT std_logic
       );
END Receiver_datapath;


ARCHITECTURE Structural OF Receiver_datapath IS
-- Components
       COMPONENT shift_register_10bit IS
              PORT(
                            CD,CLK          :       IN     std_logic;
                            D_in            :       IN     std_logic_vector(9 downto 0);
                            S                       :       IN     std_logic_vector(1
downto 0);
                            RSI,LSI         :       IN     std_logic;
                            Q                       :       OUT    STD_LOGIC_VECTOR(9
DOWNTO 0)

              );
       END COMPONENT;


       COMPONENT data_register_8bits IS
              Port (
                     CLK            :       IN     STD_LOGIC;
                     CD             :       IN     STD_LOGIC;
                  LD         :       IN     STD_LOGIC;
                     Q              :       OUT    STD_LOGIC_VECTOR(7 DOWNTO 0);
                     D_in    :       IN STD_LOGIC_VECTOR(7 DOWNTO 0)
              );
       END COMPONENT;


       COMPONENT parity_checker_9bit IS
              PORT (
                     D      :       IN     STD_LOGIC_VECTOR(8 DOWNTO 0);

                     Y      :       OUT    STD_LOGIC
              );

       END COMPONENT;

       COMPONENT counter_mod4 IS
              Port (
                     CLK    : IN    STD_LOGIC;
                     CD     : IN    STD_LOGIC;
```

```
                CE      : IN    STD_LOGIC;
                        TC4     : OUT   STD_LOGIC
                );
        END COMPONENT;

        COMPONENT counter_mod8 IS
                Port (
                        CLK     : IN    STD_LOGIC;
                        CD      : IN    STD_LOGIC;
                        CE      : IN    STD_LOGIC;
                        TC8     : OUT   STD_LOGIC
                );
        END COMPONENT;

        COMPONENT counter_mod10 IS
                Port (
                        CLK             : IN    STD_LOGIC;
                        CD              : IN    STD_LOGIC;
                        CE              : IN    STD_LOGIC;
                        TC10    : OUT   STD_LOGIC
                );
        END COMPONENT;

-- Internal wires or signals
        SIGNAL B                :       STD_LOGIC_VECTOR(7 downto 0);
        SIGNAL CE_Mod10         :       STD_LOGIC;
        SIGNAL EPB              :       STD_LOGIC;


BEGIN
-- Instantiation of components
Chip4   : counter_mod4
        PORT MAP (
-- from component name          => to signal or port name
                CLK     => CLK,
                CD      => Clear_Counters,
                TC4 => TC4,
                CE      => CE_Mod4
        );

Chip5   : counter_mod8
        PORT MAP (
-- from component name          => to signal or port name
                CLK     => CLK,
                CD      => Clear_Counters,
                TC8     => CE_Mod10,
                CE      => CE_Mod8

        );

Chip6   : counter_mod10
        PORT MAP (
-- from component name          => to signal or port name
                CLK             => CLK,
                CD              => Clear_Counters,
                TC10    => TC10,
                CE              => CE_Mod10

        );

Chip1   : data_register_8bits
        PORT MAP (
-- from component name          => to signal or port name
                D_in    => B,
                LD              => LDD,
                CD              => CD,
                CLK             => CLK,
                Q               => Rx_out
        );

Chip2   : shift_register_10bit
        PORT MAP (
-- from component name          => to signal or port name
                CLK                             => CLK,
                CD                              => CD,
                D_in                    => "0000000000",
```

```
               Q(7 downto 0)  => B,
               Q(8)                    => EPB,
               Q(9)                    => Stop_bit,
               S                               => S,
               RSI                             => Serial_in,
               LSI                             => '0'
       );

Chip3   : parity_checker_9bit
       PORT MAP (
-- from component name        => to signal or port name
               D (7 downto 0) => B,
               D(8)                    => EPB,
               Y                               => EP_check
       );


-- Extra logic in the circuit
TC8 <= CE_mod10;


END Structural;
```

Data_register_8bits.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY data_register_8bits IS
      Port (   CLK     :       IN      STD_LOGIC;
                               CD              :       IN      STD_LOGIC;
                               LD              :       IN      STD_LOGIC;
                               Q               :       OUT     STD_LOGIC_VECTOR(7 DOWNTO 0);
                               D_in    :       IN STD_LOGIC_VECTOR(7 DOWNTO 0)
       );

END data_register_8bits;

       -- Internal desciption in FSM style: Three blocks --> State register, CC1 and CC2

ARCHITECTURE FSM_like OF data_register_8bits IS

CONSTANT Reset : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00000000";

-- Internal wires --> in this case just the present and future state signals

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(7 DOWNTO 0);

BEGIN
------------------------- State register
-- The only clocked block, which is essentially a set of D-type flip-flops in parellel
-- The asynchronous reset has precedence over the CLK

State_Register: PROCESS (CD, CLK,D_in)
       BEGIN

       IF CD = '1' THEN                    --  reset  counter  (  an  asynchronous  reset
which we call "Clear Direct"
               present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
               present_state <= future_state;
       END IF;

END PROCESS State_Register;

-- CC1 ---------- Combinational circuit for calculating next state
-- Generally, even for a simple FSM, this circuit will be complicated enough
-- to use a "process" for writing it easier.

CC1: PROCESS (present_state, D_in, LD)   -- All the block inputs in the sensitivity list
       BEGIN
               IF LD='1' THEN
```

```
                        future_state <= D_in;
                ELSE
                        future_state <= present_state;
                END IF;

END PROCESS CC1;

--- CC2 ------------Combinational circuit for calculating the outputs
-- There will be circuits like this register where the implementation is simply a
buttfer
-- We are interested in buffering the internal present state

Q <= present_state;

END FSM_like;
```

Shift_register_10bit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY shift_register_10bit IS
      Port (
                CLK             : IN    STD_LOGIC;
                CD              : IN    STD_LOGIC;
                S               : IN    STD_LOGIC_VECTOR(1 DOWNTO 0);
                RSI     : IN    STD_LOGIC;
                LSI             : IN    STD_LOGIC;
                Q               : OUT   STD_LOGIC_VECTOR(9 DOWNTO 0);
                D_IN    : IN STD_LOGIC_VECTOR(9 DOWNTO 0)
       );

END shift_register_10bit;

                        -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF shift_register_10bit IS


CONSTANT Reset          : STD_LOGIC_VECTOR(9 DOWNTO 0) := "0000000000";

-- Internal wires --> in this case just the present and future state signals
SIGNAL present_state,future_state: STD_LOGIC_VECTOR(9 DOWNTO 0);

BEGIN
------------------------- State register
-- The only clocked block, which is essentially a set of D-type flip-flops in parellel
-- The asynchronous reset has precedence over the CLK
State_Register: PROCESS (CD, CLK)
        BEGIN
                IF CD = '1' THEN                        -- reset counter ( an asynchronous
reset which we call "Clear Direct"
                        present_state <= Reset;
                        ELSIF (CLK'EVENT and CLK = '1') THEN --  Synchronous  register  (D-
type flip-flop)
                                present_state <= future_state;
                END IF;

END PROCESS State_Register;

--  CC1  ----Combinational  circuit  for  calculating  next  state,  which  is  setting  the
"future_state" signal
--          Generally, even for a simple FSM, this circuit will be complicated enough
--          for using a process in order to code it easier.

CC1: PROCESS (present_state, D_IN, S, RSI, LSI)
        BEGIN
        IF S = "11" THEN
                future_state <= D_IN;
                ELSIF S="01" THEN       -- This is right shift of the putput bits; Q(0),
which is also the RSO is lost
                        future_state(0) <= present_state(1);
```

```
                        future_state(1) <= present_state(2);
                        future_state(2) <= present_state(3);
                        future_state(3) <= present_state(4);
                        future_state(4) <= present_state(5);
                        future_state(5) <= present_state(6);
                        future_state(6) <= present_state(7);
                        future_state(7) <= present_state(8);
                        future_state(8) <= present_state(9);
                        future_state(9) <= RSI;

                        ELSIF S="10" THEN -- This is left shift of the output bits; Q(9),
which is also the LSO is lost
                                future_state(9) <= present_state(8);
                                future_state(8) <= present_state(7);
                                future_state(7) <= present_state(6);
                                future_state(6) <= present_state(5);
                                future_state(5) <= present_state(4);
                                future_state(4) <= present_state(3);
                                future_state(3) <= present_state(2);
                                future_state(2) <= present_state(1);
                                future_state(1) <= present_state(0);
                                future_state(0) <= LSI;

                                ELSE
                                        future_state <= present_state;
        END IF;

END PROCESS CC1;

-- CC2 ---- Combinational circuit for calculating the outputs
Q <= present_state;

END FSM_like;
```

Counter_mod8.vhd

```
LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY counter_mod8 IS
      PORT(
         CD,CLK,CE   :   IN  std_logic;
         TC8         :   OUT std_logic
          );
END counter_mod8;

ARCHITECTURE FSM_style OF counter_mod8 IS

CONSTANT Max_Count    : STD_LOGIC_VECTOR(2 DOWNTO 0)   := "111";
CONSTANT Reset           : STD_LOGIC_VECTOR(2 DOWNTO 0)   := "000";

SIGNAL present_state, future_state   : std_logic_vector(2 DOWNTO 0);

BEGIN
----------------------------- the only clocked block : the state register
state_register: PROCESS (CD, CLK)
        BEGIN

        IF CD = '1' THEN                        -- reset counter
                present_state <= Reset;
        ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
                present_state <= future_state;
        END IF;

END PROCESS state_register;

-- CC1: combinational system for calculating next state
CS_1: PROCESS (present_state, CE)
        BEGIN
        IF CE = '1' THEN
                IF(present_state < Max_Count ) THEN
                        future_state <= present_state + 1 ;
```

```
                ELSE
                        future_state <= Reset;
                END IF;
        ELSE
                future_state <= present_state;  -- count disable
        END IF;

END PROCESS CS_1;


-- combinational logic to determine the outputs
-- CS2:
        TC8 <= '1' WHEN (present_state = Max_Count AND CE = '1') ELSE '0';


END FSM_style ;
```

Parity_generator_8bit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;

ENTITY parity_generator_8bit IS
        PORT (
                X_in    : IN STD_LOGIC_VECTOR(7 downto 0);
                EPB    : OUT STD_LOGIC

         );
END parity_generator_8bit;

-- Defining the architecture using simply logic equations derived from the truth table
-- Even parity is the function XOR

architecture logic_equations of parity_generator_8bit is

begin

   EPB   <=   X_in(0) XOR X_in(1) XOR X_in(2) XOR X_in(3) XOR X_in(4) XOR X_in(5) XOR
X_in(6) XOR X_in(7);

end logic_equations;
```

Mux4.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

ENTITY  MUX4 is

        port(   E       :       in std_logic;
                        Ch0     :       in std_logic;
                        Ch1     :       in std_logic;
                        Ch2     :       in std_logic;
                        Ch3     :       in std_logic;
                        s       :       in std_logic_vector(1 downto 0);
                        y       :       out std_logic
                );
end MUX4;

architecture truth_table of MUX4 is
begin

y       <=      Ch0 when (s ="00" and E = '1') else
                Ch1 when (s ="01" and E = '1') else
                Ch2 when (s ="10" and E = '1') else
                Ch3 when (s ="11" and E = '1') else '0';

end truth_table;
```

Counter_mod4.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY counter_Mod4 IS
      Port (
               CLK     : IN    STD_LOGIC;
               CD              : IN    STD_LOGIC;
               CE              : IN    STD_LOGIC;
               TC4             : OUT   STD_LOGIC
       );
END ;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF counter_Mod4 IS

CONSTANT Max_Count     : STD_LOGIC_VECTOR(2 DOWNTO 0) := "011"; --
CONSTANT Reset         : STD_LOGIC_VECTOR(2 DOWNTO 0) := "000";

-- Internal wires

SIGNAL present_state,future_state: STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
------------------------------ the only clocked block : the state register
state_register: PROCESS (CD, CLK)
   BEGIN

   IF CD = '1' THEN                   -- reset counter
      present_state <= Reset;
   ELSIF (CLK'EVENT and CLK = '1') THEN      -- Synchronous register (D-type flip-flop)
      present_state <= future_state;
   END IF;

END PROCESS state_register;

------------------------- ESS state_registercombinational system for calculating next
state
CS_1: PROCESS (present_state, CE)
       BEGIN
       IF CE = '1' THEN
               IF(present_state < Max_Count ) THEN
                       future_state <= present_state + 1 ;
               ELSE
                       future_state <= Reset;
               END IF;
       ELSE
               future_state <= present_state;  -- count disable
       END IF;

END PROCESS CS_1;

------------------------- CS_2: combinational system for calculating extra outputs
------------------------- and outputing the present state (the actual count)

TC4 <= '1' WHEN ((present_state = Max_count) AND CE = '1') ELSE '0';  --terminal count

END FSM_like;
```

Counter_mod10.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;
```

```
ENTITY counter_mod10  IS
      Port (   CLK     : IN    STD_LOGIC;
               CD              : IN    STD_LOGIC;
               CE              : IN    STD_LOGIC;
               Q               : OUT   STD_LOGIC_VECTOR(3 DOWNTO  0);
               TC10    : OUT   STD_LOGIC
       );

END counter_mod10;

                      -- Internal desciption in FSM style

ARCHITECTURE FSM_like OF counter_mod10 IS
-- Internal wires
                             -- State signals declaration
                             -- A BCD counter will consist of 10 diferent states
       TYPE State_type IS (Num0, Num1, Num2, Num3, Num4, Num5, Num6, Num7, Num8, Num9) ;
--------> This is important: specifying to the synthesiser the code for the states
       ATTRIBUTE syn_encoding : STRING;
       ATTRIBUTE syn_encoding OF State_type : TYPE IS "sequential";
              -- (It may be:"sequential" (binary); "gray"; "one-hot", etc.
       SIGNAL present_state, future_state   : State_type ;

              -- These two lines work OK, but the state machine is not recognised.
              --  and  regular  logic  is  synthesised  (so  it's  not  so  good  as  the
syn_encoding).
-- ATTRIBUTE enum_encoding : string;
-- ATTRIBUTE enum_encoding of State_type : TYPE IS "sequential";


-- Constants for special states (the first and the last state)
CONSTANT Reset        : State_type := Num0; -- The fisrt state. This is another name for
the state Num0
CONSTANT Max_count    : State_type := Num9; -- The last state. This is anather way to
name the state Num9

BEGIN

---------------------- State Register: the only clocked block.
---------------------- The "memory" of the system (future events will depend on past
events

state_register: PROCESS (CD,CLK, future_state)
       BEGIN

       IF CD = '1' THEN                      -- reset counter
              present_state <= Reset;
       ELSIF (CLK'EVENT and CLK = '1') THEN -- Synchronous register (D-type flip-flop)
              present_state <= future_state;
       END IF;

END PROCESS state_register;

------------------------- CC1: Combinational system for calculating next state

CC_1: PROCESS (present_state, CE)
       BEGIN
              IF CE = '0' THEN
                     future_state <= present_state;  -- count disable

              ELSE                                   -- just a simple state up count
                     CASE present_state IS
                            WHEN Num0 =>
                                   future_state <= Num1 ;
                            WHEN Num1 =>
                                   future_state <= Num2 ;
                            WHEN Num2 =>
                                   future_state <= Num3 ;
                            WHEN Num3 =>
                                   future_state <= Num4 ;
                            WHEN Num4 =>
                                   future_state <= Num5 ;
                            WHEN Num5 =>
                                   future_state <= Num6 ;
                            WHEN Num6 =>
                                   future_state <= Num7 ;
```

```
                                WHEN Num7 =>
                                        future_state <= Num8 ;
                                WHEN Num8 =>
                                        future_state <= Num9 ;
                                WHEN Num9 =>
                                        future_state <= Num0 ;
                        END CASE ;

                END IF;

END PROCESS CC_1;

------------------------- CS_2: combinational system for calculating extra outputs
------------------------- and outputing the present state (the actual count)

CC_2: PROCESS (present_state, CE)
        BEGIN
-- The terminal count output
                IF ((present_state = Max_count) AND (CE = '1') ) THEN
                        TC10 <= '1';
                ELSE
                        TC10 <= '0';
                END IF;

-- And now just copying the present state to the output:
                CASE present_state IS
                                WHEN Num0 =>
                                        Q <= "0000";
                                WHEN Num1 =>
                                        Q <= "0001";
                                WHEN Num2 =>
                                        Q <= "0010";
                                WHEN Num3 =>
                                        Q <= "0011";
                                WHEN Num4 =>
                                        Q <= "0100";
                                WHEN Num5 =>
                                        Q <= "0101";
                                WHEN Num6 =>
                                        Q <= "0110";
                                WHEN Num7 =>
                                        Q <= "0111";
                                WHEN Num8 =>
                                        Q <= "1000";
                                WHEN Num9 =>
                                        Q <= "1001";
                END CASE ;


END PROCESS CC_2;

-- Place other logic if necessary

END FSM_like;
```

Parity_checker_9bit.vhd

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;

ENTITY parity_checker_9bit IS

        PORT (
                D               : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
                Y               : OUT STD_LOGIC
        );

END parity_checker_9bit;


ARCHITECTURE logic_equations OF parity_checker_9bit IS
BEGIN
        Y <=   D(0) XOR D(1) XOR D(2) XOR D(3) XOR D(4) XOR D(5) XOR D(6) XOR D(7) XOR
D(8);
```

```
END logic_equations;
```

Anexo 3 Códigos. Top

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Top is
PORT (
              OSC_CLK_in            :       IN  STD_LOGIC;
              CD                            :       IN  STD_LOGIC;
              Enable_TX             :       IN  STD_LOGIC;
              --ST                           :       IN  STD_LOGIC;
              AG, AY, AR, BG, BY, BR, LA, LB : OUT STD_LOGIC;
              Serial_out            :       OUT  STD_LOGIC;

              --TX_IN                :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);
              Enable_RX             :       IN  STD_LOGIC;
          Serial_in          :       IN  STD_LOGIC;


              CLK_1Hz_SQUARED       :       OUT STD_LOGIC -- The 1 Hz LED to show that
the system works
       );
end Top;

architecture structural of Top is

COMPONENT UART_module IS
       PORT(
              OSC_CLK_in            :       IN  STD_LOGIC;
              CD                            :       IN  STD_LOGIC;
              Enable_TX             :       IN  STD_LOGIC;
              TX_IN                 :       IN      STD_LOGIC_VECTOR (7 DOWNTO 0);

              ST                            :       IN      STD_LOGIC;
              Serial_out            :       OUT  STD_LOGIC;

              Enable_RX             :       IN  STD_LOGIC;
          Serial_in          :       IN  STD_LOGIC;
              RX_OUT                :       OUT STD_LOGIC_VECTOR (7 DOWNTO 0);

              CLK_1Hz_SQUARED       :       OUT STD_LOGIC -- The 1 Hz LED to show that
the system works
       );
END COMPONENT;

COMPONENT traffic_control_top IS
       PORT(
       OSC_CLK_IN            : IN    STD_LOGIC;
              CLK_1Hz_LED           : OUT   STD_LOGIC;
              --CLK_2Hz_LED         : OUT   STD_LOGIC;
              N                     : IN    STD_LOGIC;
              CA, CB, PA, PB : IN    STD_LOGIC;
              AG, AY, AR, BG, BY, BR  : OUT STD_LOGIC
              --Seven_seg_digit_Unit: OUT   STD_LOGIC_VECTOR (6 downto 0);
              --Seven_seg_digit_Tens: OUT   STD_LOGIC_VECTOR (6 downto 0)
       );
END COMPONENT;

SIGNAL ST_aux         :       std_logic;
SIGNAL AG_aux         :       std_logic;
SIGNAL AY_aux         :       std_logic;
SIGNAL AR_aux         :       std_logic;
SIGNAL BG_aux         :       std_logic;
SIGNAL BY_aux         :       std_logic;
SIGNAL BR_aux         :       std_logic;
SIGNAL CLK            :       std_logic;
SIGNAL CA_aux         :       std_logic;
SIGNAL CB_aux         :       std_logic;
SIGNAL PA_aux         :       std_logic;
SIGNAL PB_aux         :       std_logic;
SIGNAL LA_aux         :       std_logic;
SIGNAL LB_aux         :       std_logic;
SIGNAL N_aux          :       std_logic;
--SIGNAL aux1         :       std_logic;
--SIGNAL aux2         :       std_logic;
```

```
SIGNAL aux3           :      std_logic;
SIGNAL aux4           :      std_logic;
SIGNAL aux5           :      std_logic;

begin
Top_1 : UART_module
       PORT MAP (
-- from component name        => to signal or port name
              CD                              =>     CD,
              Enable_TX                       =>     Enable_TX,
              OSC_CLK_in            =>     CLK,
              --ST            =>     ST,
              Serial_out                      =>     Serial_out,
              Serial_in                       =>     Serial_in,
              TX_IN(0)=>AG_aux,
              TX_IN(1)=>AY_aux,
              TX_IN(2)=>AR_aux,
              TX_IN(3)=>BG_aux,
              TX_IN(4)=>BY_aux,
              TX_IN(5)=>BR_aux,
              TX_IN(6)=>LA_aux,
              TX_IN(7)=>LB_aux,

              RX_OUT(5)=>CA_aux,
              RX_OUT(6)=>CB_aux,
              RX_OUT(4)=>N_aux,
              RX_OUT(3)=>PA_aux,
              RX_OUT(7)=>PB_aux,
              RX_OUT(2)=>aux3,
              RX_OUT(1)=>aux4,
              RX_OUT(0)=>aux5,

              CLK_1Hz_SQUARED=>ST_aux,
              ST=>ST_aux,


              Enable_RX      =>     Enable_RX
       );

Top_2 : traffic_control_top
       PORT MAP (
-- from component name        => to signal or port name
              OSC_CLK_IN                              =>     CLK,
              AG=>AG_aux,
              AY=>AY_aux,
              AR=>AR_aux,
              BG=>BG_aux,
              BY=>BY_aux,
              BR=>BR_aux,
              CA=>CA_aux,
              CB=>CB_aux,
              PA=>PA_aux,
              PB=>PB_aux,
              N=>N_aux,
              CLK_1Hz_LED=>CLK_1Hz_SQUARED

       );
CLK <= OSC_CLK_IN;
LA_aux<=BG_aux;
LB_aux<=AG_aux;
AG <= AG_aux;
AR <= AR_aux;
AY <= AY_aux;
BG <= BG_aux;
BR <= BR_aux;
BY <= BY_aux;
LA <= LA_aux;
LB <= LB_aux;

end structural;
```